Mixed Criticality Systems with Weakly-Hard Constraints

Oliver Gettings

MSc by Research

University of York

Computer Science

September 2015

Abstract

Mixed criticality systems contain components of at least two criticality levels which execute on a common hardware platform in order to more efficiently utilise resources. Due to multiple worst-case execution time estimates, current adaptive mixed criticality scheduling policies assume the notion of a low criticality mode where by a taskset executes under a set of more realistic temporal assumptions and a high criticality mode, in which all low criticality tasks in the taskset are descheduled, to ensure that high criticality tasks can meet more conservative timing constraints derived from certification approved methods. This issue is known as the *service abrupt* problem and comprises the topic of this work.

The principles of real-time schedulability analysis are first reviewed, providing relevant background and theory on which mixed criticality systems analysis is based. The current state-of-the-art of mixed criticality systems scheduling policies on uni-processor systems are then discussed along with the major challenges facing the adoption of such approaches in practice. To address the service abrupt issue this work presents a new policy, Adaptive Mixed Criticality - Weakly Hard which provides a guaranteed minimum quality of service for low criticality tasks in the event of a criticality mode change. Two offline response time based schedulability tests are derived for this model and dominance relationship proved. Empirical evaluations are then used to assess the relative performance against previously published policies and their schedulability tests, where the new policy is shown to offer a scalable performance trade-off between existing fixed priority preemptive and adaptive mixed criticality policies. The work concludes with possible directions for future research.

Contents

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor, Dr Robert I. Davis for his unrelenting support and patience during my time at the University of York.
This research was supported by EPSRC(UK) LSCITS Grant EP/F501374/1 and Rapita Systems Ltd. EPSRC(UK) Research Data Management: No new primary data created during this study.

Declaration

I declare that the work presented in this document is my own, unless explicitly indicated. This research was undertaken by myself, under the supervision of Dr Robert I. Davis at the Real-Time Systems Research Group, Department of Computer Science, University of York. This work has not been submitted for any other award at any institute.
Results in chapter 4 and chapter 5 are presented in [53]. Results in Appendix A are presented in [34, 36, 38]. The first paragraph in section 5.5 is written by Dr Robert I. Davis. All external sources are cited via the use of references.

1 Introduction

Mixed Criticality Systems (MCS) contain components of at least two criticality levels which execute on a common hardware platform. While the sharing of hardware resources may result in a more efficient implementation over traditional isolated systems, there is a significant conflict between the required certification of components and the exploitation of temporal properties to more effectively utilise the underlying platform.

High criticality components may be required by a *Certification Authority* (CA) to meet a particular standard (e.g DO-178B/C or ISO26262) which dictates the methods used to determine the timing behaviour of tasks. The *Worst-Case Execution Time* (WCET) estimate of a task determined by the methods required for certification may be overly conservative compared to the values determined by a system designer using less rigorous methods. It is this pessimism that can be exploited to more efficiently utilise hardware, provided that safeguards are in place to ensure that each component in the system is guaranteed to meet its designated level of assurance.

One way in which this issue has been approached is the concept of criticality modes. Criticality modes are ordered from the lowest to the highest level of assurance. A system starts in the lowest criticality mode with timing behaviour assumed to be that determined by the system designer. Provided that each component does not exceed its allocated execution time budget, the system will remain in this mode. However, should an over-run be detected, the system will increase the criticality mode and all components assigned a criticality below this level will either be descheduled or permitted to miss their deadlines. This form of mode change is an extreme response and despite ensuring the timing constraints determined by the certification process, could lead to a loss of functionality so severe that it is not acceptable in the design of the system.

For example, consider a unmanned aerial vehicle (UAV) which contains three components, one of High (HI) criticality (e.g the flight control system) and two of Low (LO) criticality (e.g surveillance systems). The process required for certification determines that the system utilisation of the HI criticality component is 0.6. The system designer has determined this value is 0.5 in practice. The two LO criticality

components which do not require certification are determined by the system designer to have utilisation of 0.25 each. Under the system designer's WCET assumptions, the system is schedulable with a utilisation of 1.0. Should an over-run of the HI criticality component be detected, the system will increase its criticality mode, the HI criticality component will be permitted to execute with a utilisation of up to 0.6 and the two LO criticality components will be descheduled. While this ensures that the HI criticality component meets the timing requirements needed for certification, the LO criticality components are still critical, as opposed to non-critical, and it may not be acceptable to lose their functionality completely. Furthermore, the utilisation in this HI criticality mode is now only 0.6, leaving unused system capacity in which LO criticality components could execute with a reduced Quality of Service (QoS), for example meeting $m - s$ out of m deadlines, by running surveillance jobs less frequently or by skipping s out of m jobs.

This work introduces a new approach called *Adaptive Mixed Criticality Weakly-Hard* (AMC-WH) to provide graceful degradation of low criticality tasks in the event of a criticality mode change, avoiding a complete loss of low criticality functionality.
Chapter 2 provides an introductory background to real-time systems and schedulability analysis developed for uni-processor systems. The purpose of which is to provide a solid understanding of the various scheduling techniques on which mixed criticality systems analysis is based. Chapter 3 reviews the current state of research in the field of MCS scheduling for uni-processor systems and identifies the major open issues. Chapter 4 reviews existing schedulability analysis for MCS based on fixed priority scheduling and introduces a new algorithm based on an existing policy called *Adaptive Mixed Criticality* (AMC) [14]. In Chapter 5 evaluates the relative performance of the new policy with an empirical investigation. Chapter 6 concludes with a summary and directions for future research.

2 Real-Time Systems

A real-time system is required to react to stimuli from the environment (including the passage of physical time) within time intervals dictated by the environment [79]. A classic example of a real-time system is the airbag system in a car. In the event of the collision of the vehicle, should the airbag deploy too late, the driver risks injury by colliding with the steering wheel. Alternatively, should the airbag deploy too early, the system's designed behaviour of deflating rapidly to avoid suffocation could result in it being ineffective at absorbing the impact. This is a system in which deadlines are imperative, defined as a *hard* real-time system.

Real-time systems can generally be categorised into three groups [29]:

- **Hard** : A system where a correct response must occur within a specific deadline, otherwise the system is considered to have failed.
- **Soft** : A system where response times are important but where the system will still function correctly if a deadline is missed.
- **Firm** : A system which is soft real-time but where there is no benefit from late delivery.

Due to the strict temporal nature of real-time systems, implementation and analytical techniques are needed to ensure predictability. Real-time scheduling theory is one such strategy.

Real-time scheduling theory reasons about worst-case scenarios using offline analysis to prove deadlines are met under certain conditions, modelling task interaction and temporal properties, to ensure predictable behaviour of systems.

2.1 System Model and Terminology

A system consists of a single processor executing a sporadic taskset, τ, comprising N tasks. Under a fixed priority scheduling policy, each task is assigned a unique priority, P_i, according to some policy, where $0 < P_i \leq N$.

In traditional real-time systems, each task has a number of attributes associated

with it such that $\tau_i = (T_i, D_i, C_i)$. Where T_i represents the period or minimum inter-arrival time of the task. D_i represents the relative deadline, the maximum time allowed from the task being released to completing its execution and C_i is the task's *Worst-Case Execution Time* (WCET). An invocation of a task is called a job. An unbounded number of consecutive jobs of task τ_i may be released at a maximum rate of T_i, the minimum inter-arrival time.

The worst-case *response time* of a task τ_i is denoted by R_i. This represents the maximum time from the release of the task until the completion of its execution. The notation $hp(i)$ and $lp(i)$ represent the sets of tasks in a taskset that are of higher or lower priority than τ_i respectively. B_i denotes the blocking time and is the maximum time task, τ_i, can be blocked by lower priority tasks due to contention for some resource k. Release jitter, J_i is the maximum time a job of task τ_i may be delayed after it arrives before becoming ready to execute.

The utilisation of a task τ_i is defined as $U_i = \frac{C_i}{T_i}$, while the total utilisation of a taskset, τ, is defined as $U = \sum_{i=1}^{N} (\frac{C_i}{T_i})$. It is assumed there is a discrete time model where the time granularity $\Delta = 1$; this can be considered equivalent to one processor clock cycle.

A taskset is said to be *schedulable* with respect to a scheduling algorithm if all possible valid sequences of jobs which may be generated by the taskset can be scheduled by the algorithm without missing any deadlines.
A taskset is said to be *feasible* with respect to a system such that there exists a scheduling algorithm that can schedule the taskset without missed deadlines.

Scheduling algorithm $\mathcal{A}$ is said to *dominate* scheduling algorithm $\mathcal{B}$ if all tasksets that are schedulable under $\mathcal{B}$ are also schedulable under $\mathcal{A}$ and there exists some taskset that is schedulable under $\mathcal{A}$, but not under $\mathcal{B}$.

It is also important to define the terms *Sufficient* and *Necessary* in the context of schedulability tests.

- A schedulability test is *Sufficient* if it guarantees that all deadlines for all valid sequence of jobs released by a taskset are met using a certain scheduling algorithm. However, failure of the test does not necessarily mean that the system is unschedulable.
- A schedulability test is *Necessary* if failure of the test will result in one or more deadlines of a task being missed under a certain scheduling algorithm.
- A test that is both *Sufficient* and *Necessary* is said to be *Exact*.

2.2 Fixed Priority Scheduling

Although processor scheduling originates in job-shop scheduling [5, 31], the advent of modern real-time scheduling theory is widely considered to have started with Liu and Layland's 1973 seminal paper on the subject [73].

2.2.1 Liu and Layland's Task Model

The initial task model introduced by Liu and Layland was restricted with the following assumptions [73];

(i) The requests for all tasks for which hard deadlines exist are periodic, with constant interval between requests.

(ii) Deadlines consist of run-ability constraints only, that is each task must be completed before the next request for it occurs.

(iii) Tasks are independent in that requests for a certain task do not depend on the initiation or the completion of requests for other tasks.

(iv) Run-time for each task is constant for that task and does not vary with time.

These assumptions imply that all tasks are periodic (i), that $D_i = T_i$ (ii) and all tasks are independent i.e. no resource sharing (iii). (ii) and (iv) imply that $C_i \leq D_i$ since $D_i = T_i$. It can also be derived from the paper that tasks are fully preemptible and it is assumed there are no overheads.

Although the Liu and Layland model provided a foundation on which to reason about real-time scheduling theory, the simplistic nature of the system due to the above assumptions makes it difficult to relate to real-world systems. These initial restrictions have since been relaxed as discussed in the following sections.

2.2.2 Critical Instant Theorem

The critical instant theorem presented by Liu and Layland [73], introduced the concept of a *Critical Instant*. This is a point in time when interference due to higher priority tasks is maximised. In the case of $C_i \leq D_i$, $D_i = T_i$ and no overheads, the critical instant is when a task is released simultaneously with all other higher priority tasks and those tasks are subsequently re-released as soon as possible (i.e after their period). For our model it shall be assumed the critical instant is at $t = 0$, although a critical instant may be at any point in time when all processes are simultaneously released. As this is the most difficult situation for a task to meet its deadline, only

the first absolute deadline of each task needs to be checked.

Example 1. *Consider a taskset, τ, containing two tasks, τ_1 and τ_2 where $T_1 = D_1 = 2$, $C_1 = 1$ and $T_2 = D_2 = 5$, $C_2 = 2$. If τ_1 is assigned the highest priority and τ_2 is assigned the lowest priority, the scheduled execution is as shown in Figure 2.1. As τ_1 is the highest priority task it will always meet its deadline under Liu and Layland's assumption of independent tasks. τ_2, however, needs to be checked as τ_1 may potentially cause interference, increasing its response time.*

The Critical Time Zone is the time from the critical instant ($t = 0$) to the first deadline of τ_2 ($t = 5$). As illustrated in Figure 2.1, the first release of τ_2 meets its deadline of $t = 5$ and as this greatest interference from τ_1, subsequent deadlines are guaranteed to be met (under the assumptions in subsection 2.2.1).

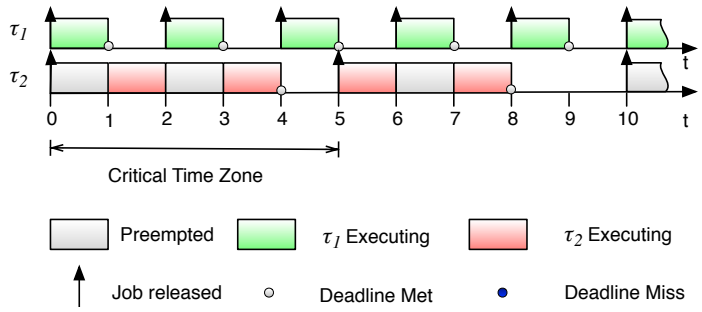

Figure 2.1: Optimal Schedule for τ_1 and τ_2

Example 2. *Consider again taskset τ, this time with the priorities of the tasks reversed where τ_2 now has the highest priority. The critical time zone is between the critical instant of $t = 0$ and the first deadline of τ_1 at $t = 2$. As τ_2 is the highest priority task, it immediately takes control of the processor preventing τ_1 from executing. At $t = 2$, τ_2 releases the processor, however this period of interference causes τ_1 to miss its deadline of $t = 2$, as shown in Figure 2.2.*
As $\forall \tau_i \mid R_i \leq D_i$ must hold for a taskset to be schedulable, this taskset is not schedulable with this new priority ordering.

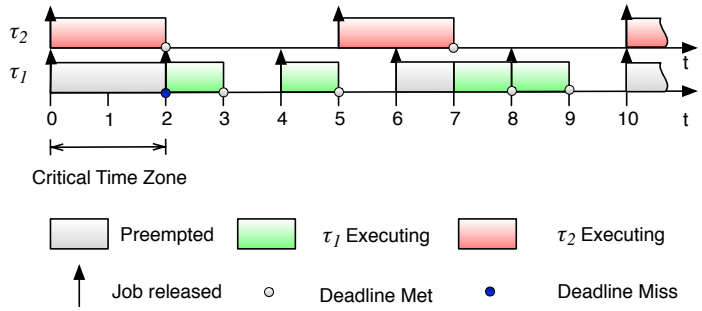

Figure 2.2: Sub-optimal Schedule for τ_1 and τ_2

2.2.3 Priority Assignment

The examples in the previous section highlight the significance that priority assignment may have on the schedulability of a taskset. Although Fineberg and Serlin (for two tasks) [48] and Wyle and Burnett (for many tasks) [94] identified that ordering tasks by their periods (shortest period $\rightarrow$ highest priority) made tasksets more likely to be feasible, it was Liu and Layland [73] who showed that *Rate-Monotonic Priority Ordering* (RMPO) was optimal for tasksets where $D_i = T_i$.

In 1982, Lung and Whitehead [70] showed that for $D_i \leq T_i$, a different type of priority assignment was optimal, named *Deadline Monotonic Priority Ordering* (DMPO) where priorities for tasks are assigned in order of relative deadlines (shortest deadline $\rightarrow$ highest priority). In 1990 Lehoczky [69] proved that neither DMPO or RMPO are optimal for arbitrary deadline tasks, where D may be greater than T, however Audsley *et al.* [2, 3] derived the *Optimal Priority Assignment* (OPA) algorithm which produces a priority ordering that results in a taskset being schedulable, if such an order exists.

Listing 2.1: Optimal Priority Assignment Algorithm

```
for each priority level k, lowest first {
    for each unassigned task τi {
        if(τi schedulable at priority k, with all other unassigned tasks
            assumed to have higher priorities ) {
            assign τi to priority k
            break (continue outer loop)
            }
     }
     return unschedulable
}
return schedulable
```

2.2.4 Schedulability Tests

While determining if a taskset is schedulable under a given algorithm is trivial for a small taskset such as that in Example 1, it can soon become intractable for large tasksets containing many tasks.

Both Liu and Layland [73] and Serlin [82] derived and proved a sufficient utilisation based test when using RMPO;

$$\sum_{i=1}^{N} \left(\frac{C_i}{T_i} \right) \leq N(2^{1/N} - 1) \tag{2.1}$$

However, Lehoczky *et al.* [68] later demonstrated that this test is not necessary (i.e pessimistic) therefore tasksets with utilisation values greater than the implied $ln(2)$ upper bound *may* be feasible. It was in this paper that Lehoczky *et al.* [68] developed an exact feasibility test for tasksets with priorities assigned using RMPO:

$$W_i(t) = \sum_{j=1}^{i} C_j \left\lceil \frac{t}{T_j} \right\rceil \leq 1 \tag{2.2}$$

$W_i(t)$ denotes the workload on the processor due to higher priority tasks, at time t which must be checked between 0 and D_i. If a value of t in this range cause the above condition to hold, then taskset is deemed schedulable.

An alternate field of schedulability analysis is *Response Time Analysis* (RTA) which was first investigated by Harter [56, 57] who developed the *Time Dilation Algorithm*. Equivalent schedulability tests were also developed by Joseph and Pandya and [62] Audsley *et al.* [4]. Audsley *et al.* [2, 3] later expanded this test to account for release jitter and blocking.

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j \tag{2.3}$$

The summation term calculates the interference due to higher priority tasks competing for the processor and preempting τ_i. As the function is non-decreasing, the equation can be solved iteratively using a recurrence relation:

$$R_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n + J_j}{T_j} \right\rceil C_j \qquad R_i^0 = C_i \tag{2.4}$$

The condition $\forall \tau_i \mid R_i \leq D_i$ must hold for the taskset to be schedulable. This schedulability test also applies to tasksets where deadlines can be less than the period of a task ($D_i \leq T_i$), therefore does not suffer the same limitation associated with Liu and Layland's utilisation based test. Davis *et al.* [33] have suggested a number of techniques to speed up convergence including new initial values.

Tindell [89] further extended this response time equation in order to support arbitrary deadlines, that is $D_i \geq T_i$ or $D_i \leq T_i$, referred to hereafter as $D_i \sim T_i$.

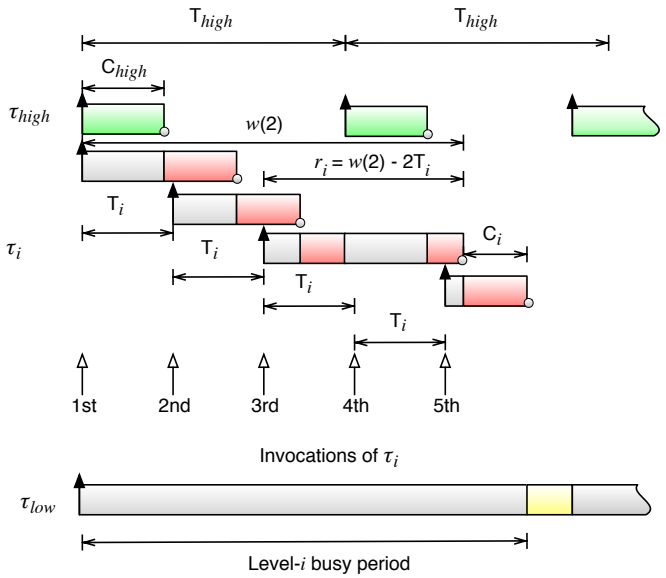

Figure 2.3: Level-i busy period over five invocations of τ_i

Using Lehoczky's notion of a level-i busy period [69], the worst case response time of a task τ_i, can be determined by assessing a number of windows. A level-i busy period is the maximum time interval for which a processor executes tasks with priority i or higher. Lehoczky proved that the longest response time for a task τ_i, is in a level-i busy period that starts at the critical instant [69].

Figure 2.3 shows an example of such a level-i busy period. With constrained deadlines ($D \leq T$), this taskset would not be schedulable, however since τ_i has a deadline that is greater than T_i, successive jobs of τ_i can be released before previous jobs have completed. All releases of τ_i in the busy period need to be assessed to determine the worst-case response time. For each overlapping release a window, q, is defined. The value $w(q)$ represents the time interval from the start of the busy

period to the completion of the current invocation, q, of τ_i (released at qT_i). This can be calculated using the recurrence relation:

$$w_i^{n+1}(q) = B_i + (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n(q) + J_j}{T_j} \right\rceil C_j \tag{2.5}$$

As the current invocation of τ_i is released at qT_i, the response time of the job is defined as the length of the window minus qT_i. Including jitter, this becomes $r_i(q) = w_i^n(q) - qT_i + J_i$. The worst-case response time of τ_i is therefore $R_i = max_{q=0,1,2...} r_i(q)$.

While each increasing value of q needs to be assessed, if a job of τ_i completes its execution *before* the next invocation, the busy period has ended and hence the worst-case response time must have been determined.

2.2.5 Shared Resources

The Liu and Layland [73] model (subsection 2.2.1) imposes a strict rule of task independence. Although this assumption aided the development of the research field in its infancy, in real systems, tasks are rarely independent and are often required to communicate.

However allowing task interaction, particularly shared resources, can lead to the blocking of higher priority tasks by lower priority tasks. This is called *priority inversion* and was first described by Lampson and Redell in 1980 [67]. An example from Burns and Wellings [29] illustrates this phenomenon well.

Example 3. *Consider the taskset below whose tasks shares resources Q and V which are accessed under mutual exclusion.*

Task	Priority	Execution Sequence	Release Time
a	4	E$Q$$Q$$Q$E	0
b	3	EE	2
c	2	E$V$$V$E	2
d	1	EE$Q$$V$E	4

Table 2.1: Execution Sequence for Taskset that exhibits Priority Inversion

Figure 2.4: Example of Priority Inversion

The lowest priority task τ_a, is the first to be released and takes control of critical section Q. τ_b and τ_c are then release at which point τ_c preempts τ_a as it is now the highest priority runnable task. At the point τ_a is preempted it still holds the lock on resource Q.

τ_c takes control of critical section V and is preempted by τ_d, also without releasing the lock it holds. τ_d executes for two cycles until it requires access to Q.

The lock to Q is held by τ_a, therefore τ_d is blocked and waits on the lock for Q. With τ_d not currently runnable it is preempted by the next highest priority task, τ_c which completes its execution and releases the lock on V.

τ_b is the next highest priority task able to execute as τ_d is still blocked waiting for resource Q. τ_b completes its execution allowing τ_a to finally complete its execution in Q to release the lock. As soon as the lock is been released, τ_d is runnable and preempts τ_a. τ_d enters Q, followed by V which was released by τ_c earlier, completing its execution. τ_a is allowed to complete its execution after τ_d has signalled completion.

Had no shared resources been involved, it would be expected that τ_d would have a worst-case response time of 4 (if shared resources were considered standard execution), instead τ_d was blocked for 13 processor cycles by lower priority tasks which were allowed to execute despite a higher priority task waiting to execute. This is called *priority inversion*.

Probably the most widely known case of priority inversion in a real system is the NASA Mars Path finder [61] which was exhibiting seemingly random resets resulting in loss of data. It was discovered, after some time, that a long running medium priority task was causing interference to a low priority task that held the mutex needed by a high priority task. This resulted in the system watchdog timer

being triggered when a higher priority data bus task missed its deadlines, causing a system reset. Fortunately the spacecraft's Real-Time Operating System (RTOS) supported the *Priority Inheritance Protocol* which was remotely activated to resolve the issue.

Priority Inheritance Protocol

Sha *et al.* [84] introduced the *Priority Inheritance Protocol* (PIP) to address the issue of priority inversion. Each task has a base priority assigned by a priority assignment protocol (e.g DMPO). If a task, τ_1, with base priority, P_0, becomes blocked by a lower priority task, τ_2, with base priority P_1, τ_2 will execute with priority P_0. If τ_2 is blocking one or more higher priority task it will execute with the highest priority of all blocked tasks. When τ_2 releases the lock on the resource its priority will return to its base value. Blocking of a task τ_i in a system were PIP is used, can be bounded [29, 84] by:

$$B_i = \sum_{k=1}^{k} usage(k, i)C_k \tag{2.6}$$

Where *usage* is a binary function, k is a critical section and C_k is the worse case execution time of the k-critical section.

Priority Ceiling Protocol

The Priority Inheritance Protocol is not without its disadvantages however. A task may be blocked multiple times and in certain situations deadlock may occur. Sha *et al.* [84] built upon the PIP to create the *Priority Ceiling Protocol* (PCP) which assigns a priority to a critical section or resource. The priority assigned to a resource, called the *priority ceiling* is equal to the highest priority of all tasks that could potentially access it.

A task, τ_i, is not able to access a resource unless its priority is greater than the ceiling priority of all resources currently locked. If the priority of the task is lower than this, it is blocked. When the task is allocated access to the critical section, it executes with its base priority unless it blocks a higher priority task, in which case it inherits the ceiling priority of the resource until it releases the lock. This new protocol not only avoids the deadlock problem of PIP, but also ensures that a invocation of a task can be blocked at most once [77].

Stack Resource Policy

Baker's [6] *Stack Resource Policy* (SRP) builds upon the Priority Ceiling Protocol with the view to avoid unnecessary context switches and can be used by both Fixed Priority and Earliest Deadline First (see section 2.3) scheduling algorithms.

The SRP assigns a preemption level p^l, to each task. In the case of fixed priority, a preemption level is equal to the static priority assigned to a task. Each resource, K, has a *Current Resource Ceiling* value $\lceil K \rceil$, which is equal to the maximum preemption level of any task that may access it. Another ceiling value called the *System Ceiling* is defined as being the maximum *Current Resource Ceiling* and is expressed as:

$$s^c = max\{\lceil K_i \rceil | i = 1, ..., m\} \tag{2.7}$$

When a task τ_i arrives it is not allowed to preempt another task until:

1. Its priority is the highest among the jobs ready to execute.
2. Its preemption level p^l is higher than *System Ceiling*.

The advantages of the the SRP is that the system may execute on a single stack and the maximum number of context switches per job is limited to two. This is due to a job having access to all the locks it requires *before* it starts execution.

2.2.6 Non-Preemptive Fixed Priority

It has been assumed up to this point that the system allows the preemption of jobs which have not yet completed their execution. An alternative approach is to allow a job to retain control of the processor until it signals completion. This method may result in less context switches at the expense of increased blocking times.

George *et al.* [52] and later Bril *et al.* [21] proved that similar to the preemptive case, the worst-case response time of a task, τ_i, is found in the level-i busy period. The subtle difference in the non-preemptive approach is that the start of the busy period is at the Δ-critical instant. A Δ-critical instant is when task, τ_i, is simultaneously released with all higher priority tasks at time t. In addition, at $t - \Delta$, a lower priority task, t_k, which has the longest execution time of all lower priority tasks, is released (should such a task exist).
Building on Tindell's [89] approach to arbitrary deadline analysis, George *et al.* developed an exact schedulability test:

$$w_i^{n+1}(q) = qC_i + \sum_{j \in hp(i)} \left(1 + \left\lfloor \frac{w_i^n(q) + J_j}{T_j} \right\rfloor \right) C_j + B_i^{NP} \tag{2.8}$$

Where

$$B_i^{NP} = \max_{\forall k \in lp(i)} (C_k - \Delta) \tag{2.9}$$

Like the preemptive version of this test, all releases of τ_i in the level-i busy period need to be assessed to find the worst-case response time.

$$R_i = max_{q=0,\ldots,Q-1}(W_i(q) - qT_i + J_i + C_i) \tag{2.10}$$

The last value of q that needs to be assessed is given by:

$$Q = \left\lceil \frac{A_i + J_i}{T_j} \right\rceil \tag{2.11}$$

Where

$$A_i^0 = \sum_{i=1}^{N} C_i \quad A_i^{j+1} = \sum_{i=1}^{N} \left\lceil \frac{A_i^j + J_j}{T_i} \right\rceil C_i \quad when \quad A_i^j = A_i^{j+1}, \quad A_i = A_i^{j+1} \tag{2.12}$$

If $\forall \tau_i \mid R_i \leq D_i$ holds, then taskset, τ, is schedulable under FP-NP. George *et al.* also demonstrated that DMPO is not optimal for constrained deadlines under FP-NP, however the Optimal Priority Assignment algorithm [2, 3] can be used to find a schedulable priority ordering, if such an order exists.

2.3 Earliest Deadline First Scheduling

Earliest Deadline First (EDF) is one of the most widely used dynamic scheduling algorithms. It assigns priority values based on a task's absolute deadline, where tasks with shorter absolute deadlines are assigned higher priorities.

2.3.1 Utilisation Test

As with fixed priority preemptive scheduling, Liu and Layland were first to introduce a feasibility test for EDF [73]. This test is exact, however it only holds for the simplified model of $D_i = T_i$.

$$\sum_{i=1}^{N} U_i \leq 1 \tag{2.13}$$

2.3.2 Processor Demand Criterion

The worst-case response time for a task under the EDF scheduling policy does not necessarily occur at the critical instant. This presents challenges for developing response time analysis similar for EDF.

In 1990 Baruah *et al.* proposed [13, 15] the *Processor Demand Criterion* (PDC), a function to calculate the load on the system due to all jobs released in time interval, t, which have absolute deadlines at or before t. This was further extended by Spuri [85] in 1996 to accommodate for jitter and blocking. The rationale is that the processor demand at any value of t must not exceed capacity; that is $\forall t \mid h(t) + b(t) \leq t$ must hold in order for a taskset to be schedulable.

The equations below can be used to calculate the processor demand and maximum blocking time respectively:

$$h(t) = \sum_{i=1}^{N} max\left(0, \left\lfloor \frac{t + J_i + T_i - D_i}{T_i} \right\rfloor\right) C_i \tag{2.14}$$

$$b(t) = max(C_{a,k} | D_a - J_a > t, D_k - J_k \leq t) \tag{2.15}$$

where $C_{a,k}$ is the maximum length of time for which task τ_a, needs to hold some resource that may also be required by task τ_k.

An upper bound for the values of t that need to be assessed is given by:

$$L_a = max\left\{(D_1 - T_1 - J_1), \ldots, (D_n - T_n - J_n), \frac{\sum_{i=1}^{n}(T_i + J_i - D_i)U_i}{1 - U}\right\} \tag{2.16}$$

This bound is not well defined for taskset where $U \approx 1$ however. To address this limitation Spuri [85] derived a second upper bound for L called the *synchronous busy period*. This is the time interval from when all tasks are released simultaneously and at their maximum rate, ending when the processor becomes idle. This value can be calculated by the recurrence equation:

$$w^0 = \sum_{i=1}^{N} C_i \qquad w^{j+1} = \sum_{i=1}^{N} \left\lceil \frac{w^j + J_i}{T_i} \right\rceil C_i \quad when \quad w^j = w^{j+1}, \quad L_b = w^{j+1} \tag{2.17}$$

The upper bound L is therefore expressed as:

$$L = \begin{cases} min(L_a, L_b) & U < 1 \\ L_b & U = 1 \end{cases} \tag{2.18}$$

Baruah *et al.* [15] observed that only a subset of values of t that correspond to absolute deadlines of jobs need to be considered, significantly reducing amount of computation required. This can be expressed logically by:

$$\forall t \mid t \in (0, L] \wedge t \in [\forall i \mid kT_i + D_i - J_i \mid k : \mathbb{N}] \tag{2.19}$$

2.3.3 Quick Processor-demand Analysis

In real systems, L can be large and so the number of deadlines that need to be assessed can become intractable. Zhang and Burns [97, 98] developed a method to reduce the number of values of t that need to be checked, called *Quick Processor-demand Analysis* (QPA). In contrast to other methods, QPA starts at L and works *backwards*, checking only absolute deadlines for tasks.

Listing 2.2: QPA Algorithm

```
t := max{d_i | d_i < L}
while (h(t) + b(t) ≤ t and h(t) + b(t) > D_min) {
    if (h(t) + b(t) < t)
        {t := h(t) + b(t)}
    else
        {t = max{ d_i | d_i < t}}
}
if (h(t) + b(t) ≤ D_min) { schedulable }
else { not schedulable }
```

The algorithm for the schedulability test is listed above, where d_i denotes the absolute deadline of a job of task τ_i. If $U \leq 1$ and $h(t) + b(t)$ converges to $\leq D_{min}$, the taskset is deemed schedulable under EDF.

2.3.4 Shared Resources

Baker's Stack Resource Policy (SRP) [6], introduced in section 2.2.5, is highlighted as being suitable for both static and dynamic scheduling algorithms to account for shared resources. Under EDF, preemption levels are assigned as the inverse of a task's relative deadline:

$$p^l(\tau_i) < p^l(\tau_j) \iff D_i > D_j \tag{2.20}$$

A task τ_i can only preempt if:

1. The absolute deadline of τ_i is the earliest deadline of all active requests in the taskset.
2. Its preemption level is higher than *System Ceiling*, defined as the highest ceiling of any resource that is currently held.

Baker also extended the EDF utilisation test to include blocking as a result of SRP [6, 97] under the condition that $D_i \leq T_i$:

$$\forall_{k=1,\ldots,n} \left(\sum_{i=1}^{N} \frac{C_i}{D_i} + \frac{B_k}{D_k} \right) \leq 1 \tag{2.21}$$

where B_k is the maximum blocking time of τ_k.

In 2015 Burns *et al.* [25] proposed an alternative approach to the SRP for EDF named Deadline-Floor Inheritance Protocol (DFP). Each shared resource is assigned a relative deadline, named the deadline-floor, which corresponds to the minimum relative deadline of all tasks that may request access to the resource.

When a task executes in a shared resource its absolute deadline, d_i, becomes the minimum of the task's current absolute deadline and the deadline-floor of the resource such that $d_i = \min\{d_i, t + DF_k\}$ where t is the time the task locks the resource and DF_k is the deadline floor of resource k. This essentially gives the executing task the highest priority of all tasks that may request control of the resource, as with Baker's SRP [6]. Furthermore, Burns *et al.* [25] show that the DFP blocking factor is indeed equivalent to SRP. The advantage of this new approach is a more simple implementation due being based on only deadlines rather than a notion of preemption levels which would require run-time monitoring.

2.3.5 Non-Preemptive EDF

In the case of non-preemptive EDF, George *et al* [52] modified Spuri's [85] processor demand function to account for the new blocking factor.

$$\forall t \leq L, \quad \sum_{i=1}^{N} max\left(0, \left\lfloor \frac{t + J_i + T_i - D_i}{T_i} \right\rfloor\right) C_i + b^{np}(t) \leq t \tag{2.22}$$

Where

$$b^{np}(t) = \begin{cases} 0 & \nexists i \ : D_i - J_i > t \\ \max\limits_{D_i - J_i > t} \{C_i - 1\} & otherwise \end{cases} \tag{2.23}$$

If Equation 2.22 holds true and $U \leq 1$, then the taskset is schedulable under EDF-NP. In addition, as blocking and jitter have already been addressed in the QPA test, substituting this revised blocking factor will result in an efficient schedulability test for EDF-NP.

2.4 Sub-optimality of Fixed Priority Scheduling

Liu and Layland proved that Earliest Deadline First Preemptive (EDF-P) is an optimal dynamic scheduling policy for implicit deadline tasksets if U $\leq$ 1 [73]. Furthermore, for a single processor, Dertouzos [42] proved that EDF-P was optimal, such that if a valid schedule exists, then it can be found under EDF-P.

Using speedup factors [63] as measure of relative effectiveness, the combined results of Liu and Layland and Dertouzos where EDF-P is considering an optimal scheduling algorithm and Fixed Priority Preemptive scheduling (FP-P) a suboptimal algorithm, the upper bound on the speedup factor is shown to be $1/ln(\Omega) \approx 1.44269$. This is the maximum value the speed of the processor must increase for a taskset that is *just* schedulable under EDF-P to become *just* schedulable under FP-P where $D = T$.

Sub-optimality of fixed priority scheduling was investigated by Davis *et al.* [40] who in 2009 proved that the upper bound on the speedup factor for FP-P vs EDF-P for constrained deadlines is $1/\Omega \approx 1.76322$. In the same year, Davis *et al.* [37] showed that for FP-P vs EDF-P, the speedup factor for arbitrary deadlines was bounded between $1/\Omega$ and 2 under optimal priority ordering [3]. Davis *et al.* [35] later found the upper bound of FP-NP vs EDF-NP to be between $1/\Omega$ and 2 for

implicit, constrained and arbitrary deadlines. In 2015 the implicit and constrained bounds were proven to be exactly $1/\Omega$ by Bruggen *et al.* [92].

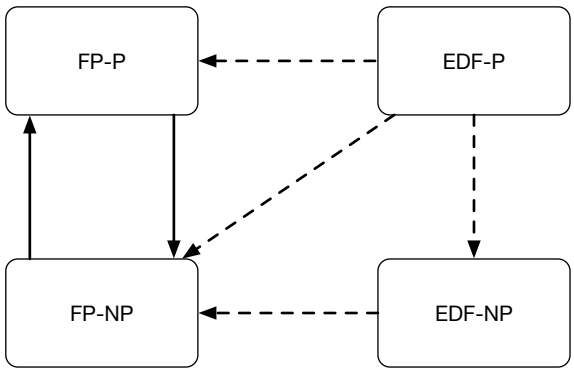

Figure 2.5: Summary of dominance relationships [38]

Figure 2.5 illustrates the relationship between EDF and FP policies where arrows denote dominance. Note that there is no dominance between FP-P and FP-NP hence both FP-P vs FP-NP and FP-NP vs FP-P can be investigated.

Using a genetic algorithm, my investigation (see Appendix A) into the sub-optimality of FP vs EDF for arbitrary deadlines discovered tasksets that were greater than the bound of $1/\Omega$ for both the preemptive and non-preemptive cases. These taskset were verified by 3 disparate implementations of the relevant schedulability tests discussed in this chapter. As a result of these findings Davis *et al.* [36] were able to derive and prove the exact bound for FP-P vs EDF-P and FP-NP vs EDF-NP for arbitrary deadlines as 2. My investigation also probed the speedup factors for FP-P vs FP-NP for implicit, constrained and arbitrary deadlines. Davis *et al.* [38] have since derived the bounds for these cases (see Table 2.2). Although Davis *et al.* [34] defined upper and lower bounds for arbitrary and constrained deadlines, results from the investigation suggest that $\sqrt{2}$ may be an exact bound. This has yet to be formally proved and remains and open problem .

For the converse case, that is FP-NP vs FP-P, Davis *et al.* [38] have proven that the exact bound for implicit and constrained deadlines is $1 + \frac{C_{max}}{D_{min}}$, where C_{max} is the maximum worst case execution time of any task in the taskset and D_{min} is the shortest relative deadline of any task in the taskset. For arbitrary deadlines this bound is $2 + \frac{C_{max}}{D_{min}}$.

Thekkilakattil *et al.* [87, 88] first investigated EDF-NP vs EDF-P, providing bounds on the speedup factor. These bounds were later tightened by Abugchem *et al.* [1].

The exact bounds were identified by Davis *et al.* [38]. Completing the model, Davis *et al.* [38] also defined the upper and lower bounds for FP-NP vs EDF-P.
The known values of upper and lower bounds on speedup factors for FP and EDF as of September 2015 are listed below.

	Lower Bound	Upper Bound
	FP-P vs EDF-P	
Implicit-deadline [73]	$1/ln(2) \approx 1.44269$	
Constrained-deadline [40]	$1/\Omega \approx 1.76322$	
Arbitrary-deadline [36]	2	
	FP-NP vs EDF-NP	
Implicit-deadline [35, 92]	$1/\Omega \approx 1.76322$	
Constrained-deadline [35, 92]	$1/\Omega \approx 1.76322$	
Arbitrary-deadline [36]	2	
	FP-P vs FP-NP	
Implicit-deadline [34]	*unknown*	$1/ln(2) \approx 1.44269$
Constrained-deadline [34]	$\sqrt{2} \approx 1.41421$	$1/\Omega \approx 1.76322$
Arbitrary-deadline [38]	$\sqrt{2} \approx 1.41421$	2
	FP-NP vs FP-P	
Implicit-deadline [38]	$1 + (C_{max}/D_{min})$	
Constrained-deadline [38]	$1 + (C_{max}/D_{min})$	
Arbitrary-deadline [38]	$2 + (C_{max}/D_{min})$	
	FP-NP vs EDF-P	
Implicit-deadline [38]	$1 + (C_{max}/D_{min})$	$2 + (C_{max}/D_{min})$
Constrained-deadline [38]	$1 + (C_{max}/D_{min})$	$2 + (C_{max}/D_{min})$
Arbitrary-deadline [38]	$2 + (C_{max}/D_{min})$	
	EDF-NP vs EDF-P	
Implicit-deadline [1, 38]	$1 + (C_{max}/D_{min})$	
Constrained-deadline [1, 38]	$1 + (C_{max}/D_{min})$	
Arbitrary-deadline [1, 38]	$1 + (C_{max}/D_{min})$	

Table 2.2: Summary of Upper and Lower Bounds on Speedup Factors

It is noted that the lower bound for the case of FP-P vs FP-NP with implicit deadlines is not known. The investigation in Appendix A has derived an empirical value of 1.34059 for this case. Further work is needed to derive a theoretical lower bound and close the gap between the empirical value and theoretical upper bound.

2.5 Summary

This chapter has reviewed techniques used for real-time systems scheduling analysis on a uni-processor system. While this is not an exhaustive review it provides the reader with enough background knowledge to grasp the concepts on which mixed criticality systems analysis is based.

Dynamic and static scheduling polices have been discussed, covering both preemptive and non-preemptive cases. Methods to handling shared resources have also been examined with particular emphasis on approaches that deal with the priority inversion problem. The chapter concludes with results that were derived from a short investigation I conducted into the sub-optimality of fixed priority scheduling listed in Appendix A. These results were then placed in the wider context of work on the sub-optimality of scheduling policies on uniprocessors systems.

3 Mixed Criticality Systems

Mixed criticality systems (MCS) consists of components of at least two levels of criticality on a common hardware platform, where criticality is the required level of assurance against failure [28]. Such systems are becoming increasingly common in the avionic and automotive electronics industry where size weight and power (SWaP) requirements are driving more efficient use of the underlaying hardware.

Due to the possible requirement for certification of HI criticality components (e.g ISO26262, DO-178B/C), there may be multiple worst case execution time estimates for the same components. This presents new scheduling challenges that can not easily be addressed by traditional real-time scheduling theory. This chapter reviews the current state of the art in terms of mixed criticality systems on uni-processor systems and discusses some of the open issues facing researchers.

3.1 System Model and Terminology

Building upon the system model outlined in section 2.1 let $\mathcal{L}$ denote an ordered, finite set of criticality levels, $\{L1, L2, ..., Ln\}$ where $L1 > L2 > \cdots > Ln$, and let L_i be the designed criticality level for task τ_i. For simplicity, this work will only consider dual criticality systems where $\mathcal{L} = \{HI, LO\}$.

The WCET value assumed for task τ_i at a specific criticality level is expressed as C_i^l where $l \in \mathcal{L}$. Vestal [91] highlighted that the WCET value is dependent on the criticality of the task; the higher the criticality the more conservative and pessimistic the estimate, that is $C_i^{HI} \geq C_i^{LO}$ for task τ_i. A task in a mixed criticality system can therefore be defined by $\tau_i = (T_i, D_i, \vec{C}_i, L_i)$ where $\vec{C}_i$ is the ordered set of C_i^l values. The utilisation of a task τ_i is defined as $U_i^l = \frac{C_i^l}{T_i}$, while the total utilisation of a taskset, τ, is defined as $U^l = \sum_{i=1}^{N} (\frac{C_i^l}{T_i})$.

R_i^{LO} denotes the worst-case response time of task τ_i in LO criticality mode whereas R_i^{HI} is the worst-case response time of task τ_i in HI criticality mode. R_i^* represents the worst-case response time of task τ_i during a criticality mode change $LO \rightarrow HI$.

Recall the set $hp(i)$ represents the set of tasks with a higher priority than τ_i. $hpHI(i)$ represents a subset of $hp(i)$ which contains tasks that are of higher priority

and higher criticality than τ_i. Similarly, $hpLO(i)$ is the subset of $hp(i)$ that contains tasks that are of higher priority and lower criticality than τ_i.

3.2 Fixed Priority Scheduling

3.2.1 The Vestal Model

Research into MCS verification was stimulated by Vestal's seminal paper [91] in 2007. Vestal outlined a task model based on the assumption that a task's WCET is dependent on the criticality level. That is, a task of higher criticality will have a larger, more conservative WCET estimate using methods appropriate for certification than if the task were of a lower criticality. This results in multiple WCET estimates for the same task, one for each criticality level.

Vestals model is a single processor system based on Fixed Priority Preemptive scheduling (FP-P) and does not account for shared resources or jitter. It is also assumed that all tasks are periodic with constrained deadlines. The notion of multiple execution time estimates for the same task, presents new scheduling challenges. A particularly crucial observation made by Vestal is that deadline monotonic (and by implication, rate monotonic) priority ordering is not optimal for the mixed criticality system model.

Example 4. *Consider a taskset from [91] shown in Table 3.1. Using traditional real-time scheduling principles, priorities would be assigned deadline monotonically*[1]*, giving task* τ_1 *the highest priority. Assuming that that the tasks do not execute for longer than their* C^{LO} *estimates the taskset is deemed schedulable.*

However if the tasks were to execute for their more pessimistic, C^{HI} *values, this priority ordering is no longer optimal and* τ_2 *would miss its deadlines due to* τ_1 *fully utilising the processor. In this case a LO criticality task has caused a HI criticality task to miss its assured deadline, a phenomenon named criticality inversion.*

Task	T	D	C^{HI}	C^{LO}	L	P
τ_1	2	2	2	1	LO	0
τ_2	4	4	1	1	HI	1

Table 3.1: Example MC Taskset

[1] Proven to be optimal for FP-P $D = T$ [70, 73]

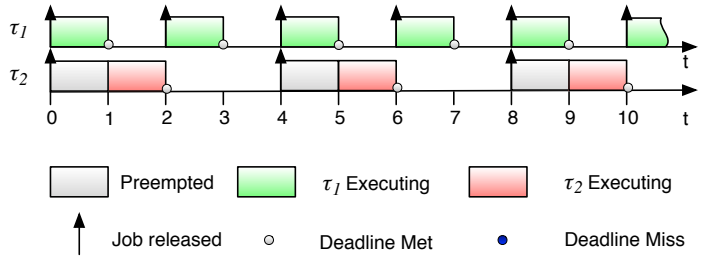

Figure 3.1: τ_1 and τ_2 executing with Criticality Level *LO*

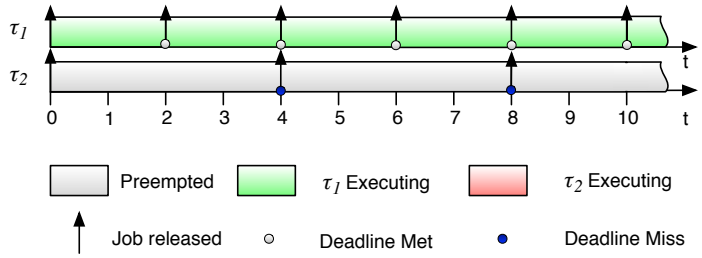

Figure 3.2: τ_1 and τ_2 executing with Criticality Level *HI*

Vestal suggested that Audsley's Optimal Priority Assignment algorithm [2, 3] may be modified to derive a priority assignment for a MCS taskset. The optimality of this approach was later proved by Dorin *et al.* [43].

One of the major limitation of the Vestal model is that the computation time of *all* tasks must be known for *all* criticality levels. This is not always possible in practice. For example, the process to obtain a WCET value for a task at the highest criticality level may involve expensive static code analysis and on-target timing measurements. It was recognised in the paper that a mechanism is required to deal with *LO* criticality jobs that attempt to exceed their designated WCET budgets to ensure that *HI* criticality tasks meet their deadlines.

Baruah and Vestal [8] later lifted the restriction of supporting only periodic task by using a sporadic task model. They compare fixed task-priority, fixed job-priority and dynamic priority schemes and present a new hybrid scheduling algorithm based on Audsley's OPA [2, 3]. The policy is shown to dominate EDF (see section 3.3) and Vestal's algorithm [91] in the sense that any taskset that is schedulable under EDF or Vestal's approach is also schedulable under this policy. It is shown to be not optimal for fixed job-priority scheduling, however, with a counter example given by the authors.

3.2.2 Period Transformation

An alternative methods to schedule mixed criticality systems is to use *Period Transformation* [83]. Using time-slicing, a HI criticality task can be scheduled as if it has a shorter period, T and execution time, C. The motivation for this is to be able to schedule a HI criticality task, which has a longer period than any LO criticality task, with a higher priority [91]. For example, should a HI criticality task have a greater period than a LO criticality task, for each HI criticality task, τ_j transform $T_j^{'} = T_j/n$ and $C^{'} = C_j/n$ for an n that results in the transformed period being less or equal to the shortest untransformed period of any LO criticality task.

Using this method in conjunction with rate monotonic priority ordering all HI criticality tasks will have higher priorities than any LO criticality tasks, avoiding criticality inversion. Tasks are in effect partitioned by criticality level, an ordering called criticality monotonic priority ordering (CrMPO).

Fleming and Burns [49] demonstrated that period transformation can be extended to many criticality levels. The increase in criticality levels complicates the analysis however. For example in a system with LO, MID and HI criticality tasks where only a MID criticality task requires transformation, it is possible for the transformed MID criticality task period to be shorter than an untransformed HI criticality task, resulting in a taskset that is not ordered criticality monotonically when rate monotonic priority ordering is applied. Instead, the process of transformation needs to be applied iteratively starting at the lowest criticality level, time-slicing the criticality level immediately above if required.

Period transformation, at least in the context of mixed criticality, is not very efficient for actual systems, however. There is an increase in context switch overheads and tasks must either be explicitly programmed to be able to be time sliced or runtime monitoring must be implemented to enforce allocated execution slots. With increasing criticality levels, the number of transformations required can also become intractable [49].

3.2.3 Job Scheduling

In 2010 Baruah *et al.* [10, 71] introduced *Own Criticality Based Priority* (OCBP), a policy based on job scheduling. A modified form of Audsley's OPA algorithm [2, 3] is used to assign priorities to jobs based the criticality of the job (rather than the criticality of the generating task). This algorithm not only gives a priority ordering but a sufficient schedulability test. The advantage of this approach is increased performance over time-partitioning, however it is not an optimal scheduling policy

as demonstrated by a given counter example. Baruah *et al.* [11] later extended this policy to many criticality levels and using speed-up factors [63] as measure of relative performance they showed that it outperformed Vestal's original analysis [91].

The reason this avenue of research was largely abandoned is that it can not be applied to recurrent tasks and there is a large amount of computation required at runtime to recalculate priorities. Guan *et al.* [55] relieved this second limitation by presenting an OCBP-based policy that limited the run-time computation of priorities to polynomial complexity. Despite this improvement, active research has favoured analysis based on recurrent tasks.

3.2.4 Static Mixed Criticality

Static Mixed Criticality (SMC) incorporates run-time monitoring to add the capability to abort overrunning *LO* criticality jobs. Presented by Baruah *et al.* [14, 26] SMC directly solves the issue of having to verify all tasks up to the highest criticality level. That is C^{HI} values for *LO* criticality tasks need not be known due to the strict enforcement of execution times.

Consider again the taskset in Table 3.1, with the SMC scheduling policy that aborts lower criticality jobs that attempt overrun. In *LO* criticality mode the schedule is the same as shown in Figure 3.1, however in *HI* criticality mode, τ_2 now meets its deadlines as any jobs of τ_1 that attempt to execute for longer than its assumed C^{LO} values will be aborted. This is illustrated in Figure 3.3 below.

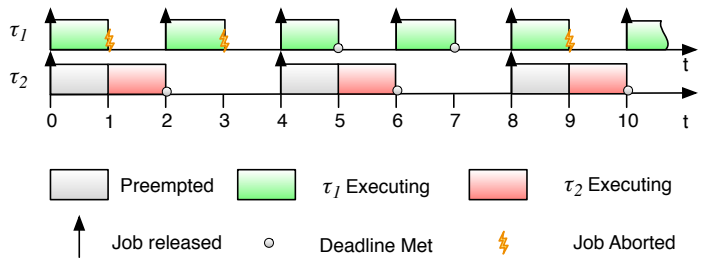

Figure 3.3: τ_1 and τ_2 executing with Criticality Level *HI* with SMC

Note that a job of τ_1 may not execute for longer than its estimated C^{LO} value. SMC enforces the execution budget by means of run-time monitoring, aborting the job should it not signal completion in time.

3.2.5 Adaptive Mixed Criticality

Baruah *et al.* [14] further utilised the notion of run-time monitoring to develop a new scheduling approach called *Adaptive Mixed Criticality* (AMC). This algorithm dominates SMC and Criticality Monotonic Priority Ordering (CrMPO) in the sense that if there is a taskset that can be scheduled by SMC or CrMPO, then it can be also scheduled by AMC.

An AMC system starts in the lowest criticality mode, LO, and jobs are assumed to execute with their up to their assumed C_i^{LO} values in order of priority. If a HI criticality job exceeds its C_i^{LO} execution time without signalling completion to the run-time monitor, then a system criticality mode change, $LO \rightarrow HI$, will occur. HI criticality tasks will be assumed to execute up to their C^{HI} values in order of priority, while all LO criticality tasks are descheduled. This is in contrast to SMC where only the overrunning LO criticality task is descheduled. While this approach allows increased schedulability of HI criticality tasks (due to slack gained from dropped LO criticality tasks), there is a complete loss of LO criticality function. Baruah *et al.* [14] presented two methods of sufficient schedulability analysis for AMC; *AMC-response time bound (AMC-rtb)* which is based on the SMC response time analysis and AMC-max which considers the points which a criticality mode change may occur and determines the maximum value to calculate the *worst-case response time*. Like SMC, priority ordering for AMC can be applied using the OPA algorithm [2, 3, 14].

Fleming and Burns [49] generalised AMC-max to n-criticality levels, however the analysis becomes computationally expensive with each additional criticality level. Carrying out analysis on the performance of n-criticality AMC, they determined that both AMC-max and AMC-rtb continue to dominate SMC and that AMC-rtb remains a good approximation to AMC-max. It can be speculated that due to this obsergation, AMC-rtb is more likely to be adopted over AMC-max, especially if the number of criticality levels increase.

Zhao *et al.* [99, 100] built upon AMC-rtb to use preemption thresholds called PT-AMC motivated by reduced stack usage. Preemption threshold scheduling (PTS) [93] allows a task to avoid preemption from higher priority task up to a bounded priority. While integrating PTS with AMC-rtb results in reduced stack usage, it is based on Baruah *et al.*'s [14] dual criticality system model and it is unclear if the sufficient schedulability tests presented will scaled to multi-criticality due to the additional complexity.

Baruah and Chattopadhyay [7] modified SMC and AMC to permit periods to vary with criticality as opposed to computation times. The advantage of this approach is

to schedule MC systems where certification pessimism is expressed in terms of task periods (in the case of event handlers), rather than WCET. This is particularly useful for communication streaming where criticality modes may correspond to levels of service.

In 2012, Davis and Bertogna [32] presented a scheme that allows tasks to have final non-preemptive regions that was shown to dominate both FP-P and FP-NP (section 2.2). Burns and Davis [24] extended AMC to use this scheme initially focusing on a dual criticality model but discussing how it can scale to multiple criticality levels. Empirical evaluations show that this scheme offers significant improvements over standard AMC in terms of schedulability. It is worth noting that this method requires RTOS support to control deferred preemption behaviour in addition to the basic run-time monitoring of AMC.

3.2.6 Zero-Slack Scheduling

An alternative approach to increasing resource utilisation of mixed criticality systems is zero-slack scheduling (ZSS). First proposed by Niz *et al.* [41], *LO* criticality tasks can be run on the slack generated by *HI* criticality tasks, when running with their C^{LO} execution budget.

The policy has two modes of operation, normal (n) and critical (c). When running in *normal* mode, tasks execute based on priority assignment. In *critical* mode, all *LO* criticality tasks are suspended to prevent interference to *HI* criticality tasks. The *HI* criticality tasks will execute in the slack created by suspended *LO* criticality jobs. The most important mechanism in zero-slack scheduling is the zero-slack instant (Z_i), this is the last possible time at which a task could switch from *normal* mode to *critical* mode to meet its deadline when they system is overloaded.

Task	T	D	C^{LO}	C^{HI}	L	P
τ_1	4	4	2	2	2	0
τ_2	10	8	2.5	5	1	1

Table 3.2: Example Zero-slack Scheduling Taskset

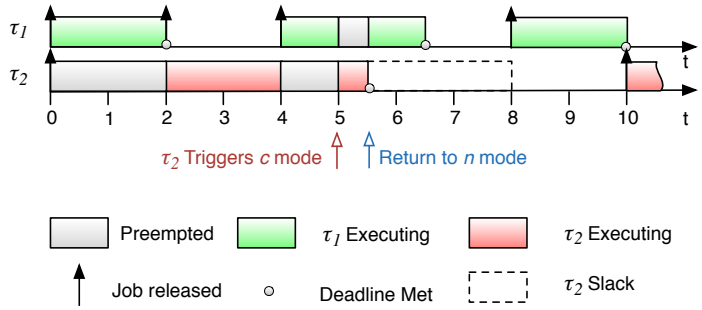

Figure 3.4: τ_1 and τ_2 under Zero-slack Scheduling Policy

Example 5. *Consider the a two task taskset with the attributes in Table 3.2 [41] whose execution is illustrated in Figure 3.4.*

At $t = 0$, τ_1 has the highest priority so gains control of the processor and completes its job at $t = 2$. τ_2 then begins to execute its job until $t = 4$ when it is preempted by a second job of τ_1. At $t = 5$, τ_2 has not completed its execution and is at its zero-slack instant (that is under overload conditions, there is just enough time for it to complete its execution and meet its absolute deadline).

The system switches to critical mode and τ_1 is suspended to allow τ_2 to complete its job. The system returns to normal mode after τ_2 has completed the remaining 0.5 units of its LO criticality budget.

The major drawback of zero-slack scheduling is that it is not designed for certification and although will execute *HI* criticality tasks at the expense of *LO* criticality tasks under overload conditions, not all *HI* criticality task deadlines can be guaranteed. Huang *et al.* [58] demonstrated an example where a *LO* criticality task overrun could cause a *HI* criticality task to miss its deadline. They also identified new analysis to incorporate this previously unaccounted interference.

While zero-slack scheduling has been shown to have low overheads [41, 58], it does not cater for sporadic tasks (which provide challenges in terms of slack) and only supports two levels of criticality (critical and normal).

3.2.7 Resource Sharing

Resource sharing is one of the most challenging issues in mixed criticality systems. While sharing of data between tasks of the same criticality is necessary, data and communication between different levels of criticality present new scheduling problems. For example a *LO* criticality task may hold the lock on a resource required by a *HI* criticality task, resulting in the *HI* criticality task missing a deadline should the *LO* execute for longer than its assumed C^{LO} value.

Burns [22] applied the original priority ceiling protocol (see section 2.2.5) to mixed criticality systems (MC-OPCP), initially focusing on dual criticality models. Resources are partitioned into groups (one per criticality level) each with their own ceiling priority. Resources can then only be locked if the resource is of the same criticality of the task trying to access it. With this protocol a task can only be directly blocked by a lower priority task of the *same* criticality level. As this approach is essentially an independent PCP scheme per criticality level, it does not allow shared resources between criticality levels. Burns made the observation that if a *LO* criticality task should be suspended (due to run-time budget enforcement) while holding the lock to a resource, it could have an impact on a future *LO* criticality job requiring the lock. To solve this problem, budget inheritance is used among *LO* criticality tasks to reduce blocking times. Although MC-OPCP was developed for dual criticality systems, due to its isolation approach, it scales well to a system with multiple criticality levels.

In 2013 Zhao *et al.* [101] developed the Highest-Locker Criticality Priority Ceiling Protocol (HLC-PCP) as an extension to AMC. Unlike MC-OPCP it is designed to allow sharing of resources between criticality levels, introducing the concept of dynamic criticality and protection for active *LO* criticality tasks that hold the lock on a resource required by a *HI* criticality task during criticality mode change. As HLC-PCP is based on the dual criticality version of AMC, it is as yet unknown the performance impact that this scheme will have on multi-criticality systems.

For the zero-slack scheduling, Lakshmanan *et al.* [66] derived two new protocols. The first, Priority-and-Criticality Inheritance Protocol (PCIP), is an extension of the Priority Inheritance Protocol (section 2.2.5) to include criticality inheritance. The inheritance of criticality prevents a task from being suspended during a criticality mode change if it is holding the lock needed by a task with a higher *base* criticality. The second protocol, named Priority-and-Criticality Ceiling Protocol (PCCP), is based on the Priority Ceiling Protocol (section 2.2.5) and provides the same advantages in the context of dual criticality zero-slack scheduling.

3.3 Earliest Deadline First Scheduling

Baruah and Vestal [8] were first to consider Earliest Deadline First (section 2.3) scheduling applied to mixed criticality systems. They noted however, that EDF was not optimal when applied to systems with tasks that have multiple execution times and so like Vestal's SMC-NO algorithm, all tasks must be verified up to the highest criticality level. This behaviour is in contrast to non-mixed criticality systems where EDF has been shown to dominate fixed priority scheduling algorithms. The authors presented a new hybrid scheduling algorithm, however it is not optimal in the sense that there are tasksets that are FJP-schedulable which are unschedulable with their policy.

Park and Kim [76] developed a dynamic scheduling algorithm for certifiable dual criticality systems called *Criticality Based EDF* (CBEDF). The algorithm is based on slack reclamation where *remaining slack* (time between C^{LO} and C^{HI} when a HI criticality task runs in LO mode) and *empty slack* (remaining time if all HI criticality tasks executes for their C^{HI} values) are given to LO criticality tasks to execute on, allowing increased utilisation. CBEDF was shown to outperform EDF and OCBP (subsection 3.2.3) in terms of schedulability, but it can not easily be extended to multiple levels of criticality, limiting its applicability to actual systems. CDEDF does have the advantage of not abandoning all LO criticality tasks in the event of a HI criticality task executing for its full C^{HI} value. However, as LO criticality jobs *only* execute on free slack (which is not fully computed until runtime) there is no control over which LO criticality jobs are abandoned in the event of a HI criticality job running for longer than its C^{LO} value.

EDF with Virtual Deadlines (EDF-VD) was introduced by Baruah *et al.* [12]. The scheme uses artificial deadlines to schedule tasks, reducing the deadline of HI criticality tasks during the LO criticality mode if the utilisation is too high (≥ 1) to schedule all tasks with their own criticality level execution times. The authors showed how this scheme may be extended to multiple criticality levels, however it suffers the limitation of other mixed criticality scheduling algorithms by dropping LO criticality tasks when a criticality change occurs. Deadlines for HI criticality tasks are also reduced by the same factor rather than on a per-task-basis.

Ekberg and Yi [45] also proposed a scheme using artificial deadlines to prioritise HI criticality tasks. HI criticality tasks are given early virtual deadlines to encourage that they execute before LO criticality tasks under EDF. In contrast to EDF-VD, deadlines are reduced on a per-task basis rather than using a global reduction factor. The motivation of this approach is that when a criticality mode change occurs, there

is less execution demand on the processor from carry over *HI* criticality jobs (active but not complete jobs) allowing them to meet their actual deadlines. The side effect of reducing the execution demand on the processor during a criticality mode change is that the demand in *LO* criticality mode is increased. Ekberg and Yi presented a pseudo-polynomial algorithm that tunes the artificial deadlines to satisfy both demand-bound functions. Their algorithm shows a large improvement over OCBP, AMC-max and EDF-VD in terms of schedulability of tasksets, however from an implementation standpoint, no evaluation of context switch overheads has been undertaken. It is unclear how much impact this would have on the schedulability on actual hardware using this algorithm. The method also suffers from the issue of abandoning all low criticality tasks after a mode change, behaviour that is not desirable on actual systems. Ekberg and Yi later generalised their approach to allow all task parameters to vary with criticality levels and indeed more than two criticality levels [46]. Their experiments showed a similar improvement to the restricted model, however as the number of criticality levels increase, the performance of this approach decreases compared to other MCS scheduling algorithms.

Easwaran [44] made the observation that Ekberg and Yi's approach was pessimistic with regards to the calculated processor demand of carry over jobs. Easwaran presents a new analysis that calculates the demand based function of *HI* and *LO* criticality jobs as a whole, providing tighter analysis and improved performance under simulation. While this algorithm offers many improvements, it is restricted to a less general task system, with only two criticality levels and constrained deadlines.

Yao *et al.* [95] also built upon Ekberg and Yi's virtual deadline scheduling algorithm. It is again restricted to constrained deadlines, however it is more easily extendable to many criticality levels than Easwaran's proposal (which becomes computationally expensive with increased levels of criticality). Yao *et al.*'s improvements come from utilising the QPA algorithm (subsection 2.3.3) for schedulability analysis named QPA-MC. This also results in less computation required for the calculations. Using a simulated annealing algorithm to assign artificial deadlines, Yao *et al.*'s approach outperforms Ekberg and Yi's algorithm in terms of schedulability of randomly generated tasksets.

A reservation-based approach was attempted by Lipari and Buttazzo [72]. In some ways it is similar to Park and Kim's CBEDF in the respect that *LO* criticality tasks run on the reclaimed processor capacity. However, Lipari and Buttazzo's approach requires a reservation server for each task that monitors execution at runtime and allocates reclaimed budget to *LO* criticality tasks when *HI* criticality tasks execute for less than their pessimistic computation values. They use a similar method to

EDF-VD to schedule HI criticality tasks as early as possible to maximise the amount of capacity reclaimed. In addition to the overheads imposed by the necessary servers, the approach does not guarantee the schedulability of low criticality tasks in low criticality mode and Lipari and Buttazzo's approach is only for dual criticality systems, limiting its application.
An interesting use of this approach is that HI criticality tasks execute independently of LO criticality task behaviours, therefore in HI criticality mode, LO criticality tasks can continue to execute in a soft-real time manner akin to a reduced (and unquantified) quality of service. This is a departure from most MCS scheduling algorithms which abandon LO criticality tasks after a criticality mode change.

Su *et al.* [86] addressed the issue of the abandonment of LO criticality tasks by using an elastic-based task model [30]. Their model allows variable periods for LO criticality tasks, with clearly defined maximum period which represents a minimum level of service. Under their Early-Release EDF (ER-EDF) scheme, if a HI criticality task executes for less than its allocated C^{HI} value, LO criticality tasks may be released earlier, executing more frequently than their maximum period without affecting the timing of the HI criticality tasks. While the advantages of this approach are clear in terms of guaranteeing a minimum frequency of service for LO criticality tasks in the HI criticality mode, it is not very general. While a minimum frequency can be guaranteed if all HI criticality tasks execute for their C^{LO} values, a *desired* frequency in LO criticality mode is not guaranteed, although the frequency is bounded to prevent a task from executing more frequently than required. It is also unclear if this scheme will scale to more than two criticality levels.

3.3.1 Resource Sharing

Zhao *et al.* [99] presented an extension to the Stack Resource Protocol (section 2.2.5) to be used with Ekberg and Yi's scheduling algorithm [45], called MC-SRP. This protocol avoids unbounded blocking by allowing a task to be blocked at most once per critical section per criticality mode. The authors go on to integrate preemption threshold scheduling [93] in their policy which they rename MC-SRPT (MC-SRP with Thresholds). This gives the additional properties of a job only being blocked before it starts execution and reduced stack usage. Although perhaps not a limitation from a safety perspective, MC-SRP(T) does not allow shared resources between criticality levels. It also only supports Ekberg and Yi's dual criticality model with constrained deadlines, rather than their more general model [46].

3.4 Criticality Modes

A common theme among most mixed criticality scheduling policies is the concept of criticality modes. A system will usually start in its lowest (least assured) criticality mode and will change modes to a higher criticality mode should tasks not behave as expected, ensuring the timing requirements of higher criticality tasks [23].

In the fixed priority scheduling policies discussed above, during a criticality change, *LO* criticality jobs are either dropped on a per task basis (SMC) or on a per criticality basis (AMC). While this ensures *HI* criticality tasks meet their timing constraints, the abandonment of *LO* critically tasks in the event of a criticality mode change is not acceptable in real systems. There needs to be a way to provide graceful degradation of *LO* criticality tasks under overload conditions to offer a reduced quality of service (QoS).

Burns and Baruah [27] proposed a revised system model that would reduce a *LO* criticality task's priority (to background level), reduce the task's execution time budget or increases the period to guarantee some level of service. The first approach allows *LO* criticality tasks to use the slack generated in the *HI* criticality mode (should any exist). The second two approaches do no take advantage of spare capacity and so require capacity reclamation methods.

Santy *et al.* [80] propose a different approach for letting some *LO* criticality tasks execute even after the system has switched to *HI* criticality mode, as long as their execution does not compromise the schedulability of *HI* criticality tasks. In effect Santy *et al.*'s algorithm delays the suspension of *LO* criticality tasks until the latest possible time rather than at the mode change, reducing the number of dropped jobs.

Jan *et al.* [60] used an elastic task model in the context of MCS by *stretching* a *LO* criticality task's period in order to decrease the load on the system in *HI* criticality mode. This permits *LO* criticality functionality with a reduced frequency at the expense of using online-decision algorithms to compute the required *stretch factor*.

In 2014, Fleming and Burns [50] introduced the notion of *importance* in mixed criticality systems, allowing the system designer to specify which *LO* criticality tasks are suspended first in the event of a criticality mode change. This provides more control over how the system should degrade. Fleming and Burns also highlighted that this approach could be used to group a number of *LO* criticality tasks together (as applications), providing a more realistic system model. The investigation was limited to a dual criticality model however and does not allow dropped low criticality tasks to be reintroduced, although this would be the next logical extension to their scheme.

Gu *et al.* [54] presented a policy where the system is isolated into components with each assigned a tolerance value. If the number of HI criticality tasks executing with their HI criticality behaviour within a component does not exceed this tolerance, than the criticality mode change within the component has no effect on the schedulability of LO criticality tasks executing in other components.

For EDF, Su *et al.* [86] have tried to address the issue of abandoning tasks by also using an elastic mixed-criticality task model, introducing a policy called Early-Release EDF (ER-EDF) which allows LO criticality tasks to be released early on the slack generated by HI criticality tasks. This variable period allows LO criticality tasks to be more easily schedulable although, as discussed in section 3.3, while a minimum service for LO criticality tasks can be guaranteed, a maximum service can not, limiting its usefulness.

The ability to return to a system's initial, LO criticality mode is desirable, not only to return to more realistic execution budgets for HI criticality tasks, but to reinstate LO criticality tasks that may have been descheduled during a criticality mode change. The most obvious choice would be to return to LO criticality mode when the system becomes idle [90]. This is in fact what most mode changing systems do. Burns and Baruah's [27] execution budget approach for example allows all tasks to have their execution budgets returned to their C^{LO} values if an idle tick is detected, as does Santy *et al.* [80].

Santy *et al.* [81] later proposed the Safe Criticality Reduction (SCR) scheme that allows a mixed criticality system to return to LO criticality mode without an idle tick (such as with multiprocessor systems). This scheme iteratively checks which jobs become inactive, starting with the highest priority HI criticality job. Once the lowest priority, HI criticality job is inactive, the system can safely switch to LO criticality mode and reintroduce LO criticality tasks.

Erickson *et al.* [47] approached the the problem from the perspective of their mixed criticality multicore framework (MC^2). Using a virtual-time mechanism to alter the period of lower criticality tasks, they demonstrated the ability to provide a scalable recovery time from overload conditions without the complete loss of all LO criticality functionality.

In 2015 Bate *et al.* [17] developed a *Bailout Protocol* for mixed criticality systems which provides a facility to return to LO criticality mode. The protocol enters a *bailout* mode if a HI criticality task executes for its assumed C^{LO} value without signalling completion. The overrunning HI criticality task is granted a computation *loan* to allow execution up to its C^{HI} value. Slack generated by the abandonment of LO criticality jobs and HI criticality tasks executing for less than their assumed C^{HI}

value is used to replenish the notional computation *debt*. When the bailout fund is zero and the lowest priority HI criticality job has completed its execution, it is safe to return to LO criticality mode. Furthermore, Bate *et al.* [17] demonstrated that by using offline sensitively analysis to tune assumed C^{LO} values of tasks, it is possible to reduce the number of times a system will enter the *bailout* mode.

3.5 Implementation

Implementation research for mixed criticality system is vast and worthy of its own literature review, therefore only some of the issues directly related to the implementation of previous discussed scheduling theory shall be touched upon.

Run-time monitoring support is assumed for many of the scheduling polices to enforce execution budget such as with SMC and AMC but such behaviour may not be natively supported by the host real time operating system. Baruah and Burns [9] illustrated how a dual criticality system could be implemented in Ada 2005[2]. The system is based on SMC and supports run-time monitoring, resource sharing (via protected objects), criticality mode change, and the ability to return to LO criticality mode when the system becomes idle. The implementation assumes only a single processor system but does illustrates a mixed criticality model that may not require modification to the host real-time OS.

Kim and Jin [64] implemented a dual criticality framework on the eCos [74] RTOS and were able to measure the performance on actual hardware. Interestingly, Kim and Jin's model allows tasks to change criticality mode independently of each other and only change modes when a certain threshold of overrun has been detected. The motivation for this is higher utilisation of hardware, although it could be argued that allowing HI criticality tasks to miss a number of deadlines is inherently dangerous and would go against certification standards. Regardless, the implementation does provide a good proof-of-concept and Kim and Jin's inclusion of (m,k)-firm deadlines [78] gives the opportunity to tune (offline) a minimum quality of service for tasks under overload conditions before a mode change is required.

Niz *et al.* [41] have implemented their zero slack scheduling approach on Linux/RK [75], a kernel that allows partitioning of resources. The overheads introduced by reservation servers are not evaluated however and the implementation only supports two criticality levels, a limitation of the underlying theory.

Zimmer *et al.* [102] have developed a processor with native hardware support for mixed criticality systems. This removes the issue of having to certify the RTOS,

[2] http://www.adacore.com

but sacrifices are made in terms of the guarantees of schedulability of *LO* criticality tasks. The processor runs hard real-time tasks where deadlines are guaranteed to be met and soft tasks which run on the spare processor capacity which may not be known until run-time.

Huang *et al.* [58, 59] investigated and compared the overheads of user-space implemented zero-slack scheduling, SMC-NO, period transformation and AMC. They concluded that AMC performs the best among fixed priority scheduling polices for mixed criticality systems.

3.6 Summary

Mixed criticality systems present a number of new research issues and challenges. This chapter provides a high level overview of the state-of-the-art of research into mixed criticality systems on uni-processor systems, covering both fixed priority and dynamic scheduling policies along with proposed shared resource techniques.

The open issues facing the adoption of the discussed mixed criticality system scheduling policies in real-world applications are numerous, however particular attention is drawn the abandonment issue of *LO* criticality tasks in the event of a criticality mode change. It is this issue that is the focus of the next chapter.

4 Mixed Criticality Systems with Weakly-Hard Constraints

One of the major issues hindering the real-world application of AMC is abandonment of all *LO* criticality jobs in the event of a criticality mode change. The abandonment problem of MCS scheduling has been addressed in various ways as discussed in chapter 3, however previous methods aimed at allowing *LO* criticality tasks to execute after a criticality mode change have mostly been best effort with limited or no guarantees over the level of service provided.

For traditional (single criticality level) real-time systems there is a concept called *weakly-hard* that could help provide stronger guarantees. The motivation behind weakly hard real-time systems scheduling is to address the disparity between hard real-time tasks and soft real-time tasks, in that hard real-time tasks are required to meet all their deadlines where as soft real-time tasks are permitted to miss deadlines but where the number of missed deadlines are not quantified. Weakly hard real-time tasks are in fact permitted to miss some deadlines however the number of missed deadlines must be *strictly bounded*.

In 1995 Hamdaoui and Ramanathan [78] introduced (m,k)-firm deadlines for streams where at least m deadlines in k consecutive invocations must be met. Priorities are dynamically assigned to streams based on the number of recently missed deadlines relative to their (m,k)-firm values. However, their (m,k)-firm scheduling algorithm does not guarantee a minimum level of service and all streams are assigned a global (m,k) value. Also in 1995, Koren and Shasha [65] worked on a different approach called *skip factor* where, when a system is overloaded, one in s invocations of a task are dropped (skipped). Bernat *et al.* [18, 19] later presented a model which in some ways is similar to the mixed criticality model of mode changes. Here, a system runs under one set of scheduling assumptions, but reverts to a *panic mode*, which guarantees that deadlines are met according to some prior offline analysis, should deadlines be at risk of being missed.
The result of the weakly-hard approach is that higher utilisation of resources is possible due to pessimism in scheduling assumptions (such as WCET) while tasks

are guaranteed to meet a minimum level of service under overload conditions. Bernat *et al.* [18] made the important observation that the order of missed deadlines may be as important as the number of missed deadlines. For example consecutive deadlines misses in an audio-visual system may have a more significant effect on functionality than if missed deadlines were distributed more evenly. While the advantages of weakly-hard are clear, the offline analysis for Bernat's model is complex as the entire hyper-period[1] of a taskset needs to be assessed.

A guaranteed reduced Quality of Service (QoS) for LO criticality tasks in a HI criticality mode would be highly desirable in addressing the abandonment issue. While weakly-hard analysis can become complex, imposing the restriction of allowing only consecutive skips in a fixed cycle prevents the problem from becoming intractable. In this work weakly-hard constraints are incorporated into the existing AMC [14] scheduling policy to provide graceful degradation of LO criticality tasks in HI criticality mode by ensuring that they meet $m - s$ out of s deadlines, skipping m to relieve the load on HI criticality tasks.

4.1 Existing Analysis

This section reviews existing analysis for MCS policies based on fixed priority scheduling, in particular recapitulation of the analysis for AMC-rtb and AMC-max since this work later builds on these to provide analysis for the AMC-WH policy.

4.1.1 Fixed Priority Preemptive Scheduling

Fixed Priority Preemptive Scheduling (FPPS), reviewed in chapter 2, is an established scheduling policy for real-time systems. Formal analysis was first developed by Liu and Layland [73]. Response time analysis was later developed by Joseph and Pandya [62] and Audsley *et al.* [4] under the assumption of a sporadic task model.

$$R_i = C_i^{L_i} + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j^{L_j} \tag{4.1}$$

Equation 4.1 can be used to calculate the worst-case response time for each task in a mixed criticality environment. This form of analysis assumes that the host Real-Time Operating System (RTOS) supports run-time monitoring to prevent LO criticality jobs from executing for longer than their C^{LO} values. For the taskset to be schedulable the following condition must hold, $\forall \tau_i : R_i \leq D_i$.

[1] The hyper-period is the least common multiple of all task periods.

While Deadline Monotonic Priority Ordering (DMPO) has been proven optimal for uniprocessor fixed priority preemptive systems [73], Vestal showed that it is not optimal for MCS [91] in the case where run-time monitoring is not supported. This is due to an issue where a higher priority, *LO* criticality job executing for more than its assumed C^{LO} value may prevent a *HI* criticality task from executing, causing it to miss its deadline. This is referred to as criticality inversion and highlights the conflict between criticality (functional importance) and priority (scheduling importance).

4.1.2 Criticality Monotonic Priority Ordering

To address the challenges of scheduling tasks with multiple WCET estimates and the issue of criticality inversion, Criticality Monotonic Priority Ordering (CrMPO) was devised. Tasks are first partitioned by criticality and then ordered by DMPO within criticality levels. This results in higher criticality tasks being prioritised over lower criticality tasks. Equation 4.1 can be used to perform response time analysis, where each task assumes the execution time for its designated criticality level.

4.1.3 Static Mixed Criticality - NO

Static Mixed Criticality - No Run-time Support (SMC-NO) is the name given by Baruah *et al.* [14] to Vestal's original analysis for MCS [91]. As the analysis assumes no run-time support, all higher priority, *LO* criticality tasks need to be verified up to the highest criticality level of any task to which they may cause interference.

$$R_i = C_i^{L_i} + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j^{L_i} \tag{4.2}$$

Equation 4.2 is used to perform response time analysis. Note that the WCET value for an interfering task τ_j is the same level as the task being assessed, L_i, rather than L_j as in (4.1). Priority assignment for SMC-NO is performed using Audsley's Optimal Priority Assignment (OPA) algorithm [2, 3], which was proved to be optimal for SMC-NO by Dorin *et al.* [43].

4.1.4 Static Mixed Criticality

Static Mixed Criticality (SMC) is an extension of Vestal's analysis [91] by means of run-time monitoring of task execution times [14, 26]. If a *LO* criticality job attempts to execute for longer than its C^{LO} budget, it is either aborted or suspended. The response time equation of Vestal's SMC-NO is modified to become:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \min(C_{jL_i}, C_{jL_j}) \tag{4.3}$$

For two tasks, τ_i, τ_j where $P_i < P_j$, the value of C_j used in the response time analysis equation depends on the three cases below:

1. If $L_i = L_j$, use C_j.
2. If $L_i < L_j$, the lower value $C_j^{L_i}$ should be used as this is the level of criticality that the task needs to be verified to.
3. If $L_i > L_j$, there is criticality inversion and so C_j should be used but with run-time system monitoring to abort if an overrun is detected.

The use of these rules ensures that only the criticality levels up to the assurance level, L_i, of a task need to be assessed. The result is that, unlike Vestal's original approach, *LO* criticality tasks do not need to be verified to the highest criticality level of the system nor do *HI* criticality WCET values for *LO* criticality tasks need to be known. As with Vestal's version of SMC, priorities are assigned using Audsley's OPA algorithm [2, 3].

4.1.5 Adaptive Mixed Criticality - rtb

Adaptive Mixed Criticality (AMC) builds upon the SMC [14] notion of run-time monitoring. That is if a job of a *HI* criticality task does not signal completion by its allocated C^{LO} budget, then a criticality mode change will occur. Further *LO* criticality jobs are descheduled and *HI* criticality jobs are assumed to execute for at most their C^{HI} values. Baruah *et al.* [14] developed two sufficient schedulability tests for this policy, the first being AMC - response time bound (AMC-rtb). Equation 4.4 is used to assess all tasks, using their C^{LO} values in *LO* criticality mode. The condition $\forall \tau_i \mid R_i^{LO} \leq D_i^{LO}$ must hold for the system to be schedulable in *LO* criticality mode.

$$R_i^{LO} = C_i^{LO} + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{LO}}{T_j} \right\rceil C_j^{LO} \tag{4.4}$$

Equation 4.5 considers only *HI* criticality tasks using their C^{HI} values. Recall that *hpHI(i)* is the set of *HI* criticality tasks with higher priority than τ_i. The condition $\forall \tau_i : L_i = HI \mid R_i^{HI} \leq D_i$ must hold true for the system to be schedulable in *HI*

criticality mode.

$$R_i^{HI} = C_i^{HI} + \sum_{j \in \mathbf{hpHI}(i)} \left\lceil \frac{R_i^{HI}}{T_j} \right\rceil C_j^{HI} \tag{4.5}$$

The analysis to assess the schedulability of the criticality change is a little more complex. Since a change in criticality for a HI criticality task τ_i must occur before R_i^{LO}, the interference from higher priority, LO criticality tasks (*hpLO(i)*) is bounded, as after this time LO criticality jobs would be descheduled. R^* is the response time of a HI task during a criticality change ($LO \rightarrow HI$). The first summation term represents the interference from higher priority, HI criticality tasks. The second summation term is interference from higher priority, LO criticality tasks that arrive *before* the criticality change (i.e before R_i^{LO}).

$$R_i^* = C_i^{HI} + \sum_{j \in \mathbf{hpHI}(i)} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j^{HI} + \sum_{k \in \mathbf{hpLO}(i)} \left\lceil \frac{R_i^{LO}}{T_k} \right\rceil C_k^{LO} \tag{4.6}$$

Audsley's OPA algorithm [2, 3] can be used to find a priority ordering that allows a taskset τ to be schedulable according AMC-rtb analysis, if such a priority ordering exits.

4.1.6 Adaptive Mixed Criticality - max

AMC-rtb suffers from pessimism when considering the mode change, since it assumes that all jobs of HI criticality tasks up to time R^* may execute with their C^{HI} values.

Initially, a HI criticality job may execute in LO criticality mode. During its execution, a criticality mode change may occur and so the job must be assumed to execute up to its C^{HI} value. As it is not known exactly when the mode change will occur, AMC-rtb pessimistically assumes the worst-case, is that all jobs of all HI criticality tasks execute with their C^{HI} values before and after the criticality change. The AMC-max analysis removes this pessimism [14] by observing that there is a bounded interval in which a criticality change may affect the response time of a HI criticality job of task τ_i. That is between 0 and R_i^{LO}. If a criticality change occurs after R_i^{LO} then the job has already completed its execution and so will not be affected by the mode change. Further invocations of the task in HI criticality mode can be verified the same as AMC-rtb using (4.5).

Let y represent the time of the criticality mode change. Figure 4.1 illustrates a criticality mode change at time y, affecting a HI criticality task τ_i under AMC-max

scheduling analysis assumptions. Note that if a criticality change is signalled while a job of τ_i is executing, it will be assumed to execute with its C_i^{HI} value.

$$R_i^y = C_i^{HI} + \sum_{k \in \mathbf{hpLO}(i)} \left(\left\lfloor \frac{y}{T_k} \right\rfloor + 1 \right) C_k^{LO} \; + \sum_{j \in \mathbf{hpHI}(i)} \left(M(j, y, R_i^y) C_j^{HI} + \left(\left\lceil \frac{R_i^y}{T_j} \right\rceil - M(j, y, R_i^y) \right) C_j^{LO} \right) \tag{4.7}$$

$$M(j, y, t) = min \left\{ \left\lceil \frac{t - y + D_j}{T_j} \right\rceil, \left\lceil \frac{t}{T_j} \right\rceil \right\} \tag{4.8}$$

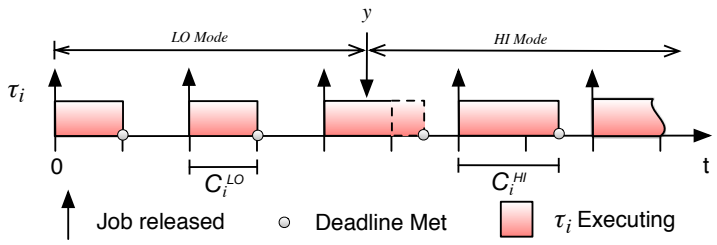

Figure 4.1: Criticality mode change under AMC-max

The worst-case response time of HI criticality task τ_i is calculated assuming interference from higher priority, LO criticality tasks released *before* the criticality change, y, plus interference from higher priority, HI criticality jobs *active at or after* the criticality change executing with up to their C^{HI} values and those completing *before* the criticality change with their C^{LO} values. Equation 4.8 calculates the maximum number of releases of a task, τ_j, after the criticality change occurs at y, up to time t [14].

The values of y that need to be assessed are bounded by 0 and R_i^{LO}, however the number of values can be large. Baruah *et al.* [14] observed that the points at which y may affect the response time of a task, correspond to the releases of higher priority, LO criticality tasks. The worst case response time of a HI criticality task during a criticality mode change is therefore given by: $R_i^* = max(R_i^y) \forall y$ where $y \in kT_j \mid \forall j \in hpLO(i) \; \wedge \; y \leq R_i^{LO} \mid \forall k : \mathbb{N}$.

4.2 Adaptive Mixed Criticality - Weakly Hard

This section introduces Adaptive Mixed Criticality - Weakly Hard (AMC-WH). This new scheduling policy allows a number of consecutive jobs of LO critically tasks to be skipped when in HI criticality mode. This reduces the load on the system, freeing up capacity for HI criticality tasks while also providing a degraded service for LO criticality tasks, which are guaranteed to meet $m - s$ out of m deadlines, where s is the number of skips and m is the length of the cycle.
The number of skips permitted and the number of subsequent deadlines that must be met ($m - s$) may be a requirement from the design of a control algorithm [51] or it may derive from physical properties of the system, for example with a radar altimeter it may be acceptable to drop some readings, but not to lose them altogether.

As an illustrative example consider Figure 4.2 which describes a LO criticality task initially executing in LO criticality mode. Task τ_k has been assigned the weakly-hard constraints that state that it must skip every 2 consecutive jobs over every 4 releases. Upon entering the HI criticality mode, the task observes these constraints, starting skipping at the first release in HI criticality mode. This cycle of skipping will repeat indefinitely providing the system remains in HI criticality mode.

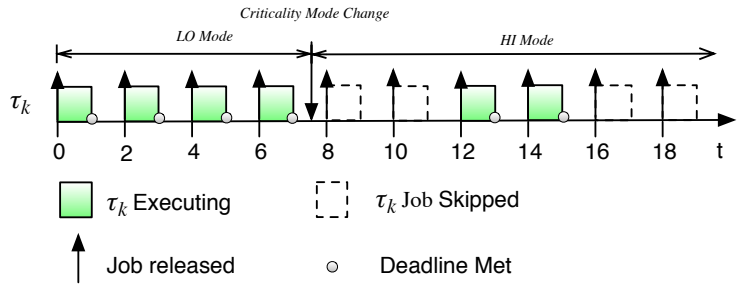

Figure 4.2: Example AMC-WH Execution

Building on the model in section 3.1, let s_k equal the number of skips assigned for the task τ_k and m_k equal the assigned cycle length in task periods. Let n equal the position of a skipped job of task τ_k from the end of the cycle such that the release of a skipped job is at $m_k T_k - nT_k$.
In the transition from $LO \rightarrow HI$ criticality, the jobs of LO criticality task τ_k released before the mode change will continue to completion as assumed with AMC, however the next release of τ_k will be the start of the consecutive skips s_k in the cycle m_k (as shown in Figure 4.3).

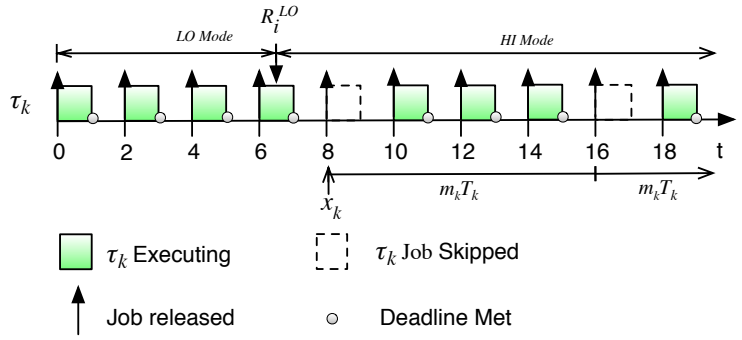

Figure 4.3: Criticality Change of τ_k

The maximum amount of execution of the task τ_k in an interval of length t (see Figure 4.4) can be expressed as the the number of jobs of τ_k assuming no skips, minus the number of skipped jobs in each cycle. Equation 4.9 computes this value for any value of n such that $1 \leq n \leq m_k$.

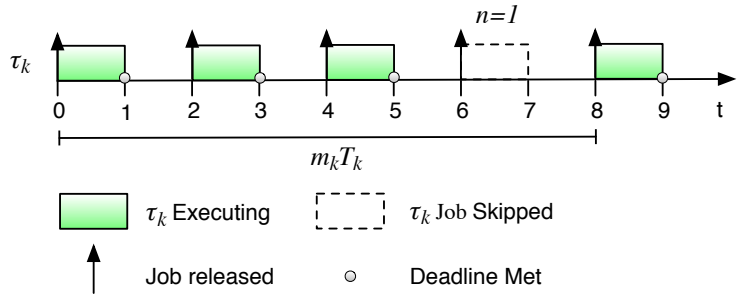

Figure 4.4: Cycle of τ_k

$$\left(\left\lceil \frac{t}{T_k} \right\rceil - \left\lceil \frac{t-(m_k-n)T_k}{m_kT_k} \right\rceil\right) C_k \tag{4.9}$$

As n represents the position of a skipped job in a cycle, (4.9) can be generalised to account for a number of consecutive skips s_k, where $1 \leq s_k \leq m_k$.

$$\left(\left\lceil \frac{t}{T_k} \right\rceil - \sum_{n=1}^{s_k} \left\lceil \frac{t-(m_k-n)T_k}{m_kT_k} \right\rceil\right) C_k \tag{4.10}$$

Note that the worst-case execution in an interval of length t occurs when the phasing of consecutive skips is at the end of a cycle i.e $n = 1, 2.., s_k$.

4.2.1 AMCrtb-WH

The susequent subsections extend the AMC schedulability analysis (subsection 4.1.5) to account for these weakly hard constraints as follows.

1) Schedulability of the *LO* Criticality Mode

In *LO* criticality mode, the AMC-WH model behaves the same as AMC. *HI* criticality and *LO* criticality tasks are assumed to execute with their C^{LO} values and so the worst-case response time of each task can be calculated using (4.4).

2) Schedulability of the *HI* Criticality Mode

The worst-case response time occurs when skips s_k, are at the end of the cycle m_k for each higher priority, *LO* criticality task τ_k. Therefore in the *HI* criticality mode, the worst-case response time of a task τ_i can be expressed as its computation time, plus the interference from higher priority, *HI* criticality tasks executing with their C^{HI} values, plus the interference from higher priority, *LO* criticality tasks, minus the interference from skipped jobs hence;

$$R_i^{HI} = C_i^{L_i} + \sum_{j \in \mathbf{hpHI}(i)} \left\lceil \frac{R_i^{HI}}{T_j} \right\rceil C_j^{HI} \; + \\ \sum_{k \in \mathbf{hpLO}(i)} \left(\left\lceil \frac{R_i^{HI}}{T_k} \right\rceil - \sum_{n=1}^{s_k} \left\lceil \frac{R_i^{HI} - (m_k - n)T_k}{m_k T_k} \right\rceil \right) C_k^{LO} \tag{4.11}$$

where $s_k < m_k$.

Unlike AMC, both *HI* and *LO* criticality tasks need to be assessed using the above analysis. In the event that 100% skipping for a *LO* criticality task is used (i.e $s_k = m_k$), the *LO* criticality task in question does not need to be assessed. In addition the task will produce zero interference to lower priority tasks in *HI* criticality mode.

3) Schedulability of the Criticality Mode Change

Consider Figure 4.3 which shows the execution of a *LO* criticality task τ_k. If a *HI* criticality task τ_i reaches its R_i^{LO} without signalling completion then a criticality mode change will be triggered. If there is a job of τ_k executing at this time then it will be allowed to complete; however the next s_k releases of τ_k, starting at time x_k, will be skipped. This is in contrast to standard weakly-hard [18] systems and is aimed at increasing schedulability during the criticality mode change.

Equation 4.6 is modified to become (4.12) to assess the schedulability of the mode change for HI criticality tasks. The worst-case response time includes the interference from higher priority, HI criticality tasks, assuming C^{HI} values for all releases from $t = 0$ to R^*, plus interference from higher priority, LO criticality tasks, minus the interference from skipped jobs between x_k and R^*.

$$R_i^* = C_i^{HI} + \sum_{j \in \mathbf{hpHI}(i)} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j^{HI} \; + \\ \sum_{k \in \mathbf{hpLO}(i)} \left(\left\lceil \frac{R_i^*}{T_k} \right\rceil - \sum_{n=s_k}^{m_k} \left\lceil \frac{R_i^* - (m_k - n)T_k - x_k}{m_k T_k} \right\rceil_0 \right) C_k^{LO} \tag{4.12}$$

where $x_k = \left\lceil \frac{R_i^{LO}}{T_k} \right\rceil T_k$ and $s_k < m_k$.

Note that following the criticality mode change, the phasing of skips of LO criticality jobs occurs at the beginning of the cycle, hence $n \in [s_k, m_k]$ is used in the summation term, rather than $n \in [1, s_k]$. During the fixed point iteration, when $R^* < x_k$, the use of $\lceil a \rceil_0$ denoting $max(\lceil a \rceil, 0)$ lower bounds $\lceil a \rceil$ by 0 is used to avoid including a negative number of skips.

Now consider the schedulability analysis for LO criticality tasks across the criticality mode change. As it is unknown when the criticality mode change may occur, the worst-case response time for LO criticality tasks may be upper bounded using (4.13) which assumes that there are no skips of LO criticality jobs up to R^*.

$$R_i^* = C_i^{LO} + \sum_{j \in \mathbf{hpHI}(i)} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j^{HI} + \sum_{k \in \mathbf{hpLO}(i)} \left\lceil \frac{R_i^*}{T_k} \right\rceil C_k^{LO} \tag{4.13}$$

4.2.2 AMCmax-WH

For a HI criticality task τ_i, AMCrtb-WH is pessimistic with regards to the mode change due to not only assuming that all higher priority, HI criticality tasks execute with their C^{HI} values up to R_i^* (4.12), but also that there is no skipping of LO criticality jobs up to R_i^{LO} (4.4).

Using the same principles as AMC-max, AMCmax-WH addresses this pessimism by taking into account the points at which a criticality change may occur. Figure 4.5 illustrates how a criticality mode change at time y may affect a LO criticality task (Figure 4.1 shows how a HI criticality task may be affected). LO criticality mode and HI criticality mode schedulability for all tasks can be assessed using the same approach as AMCrtb-WH (see (4.4) and (4.11)).

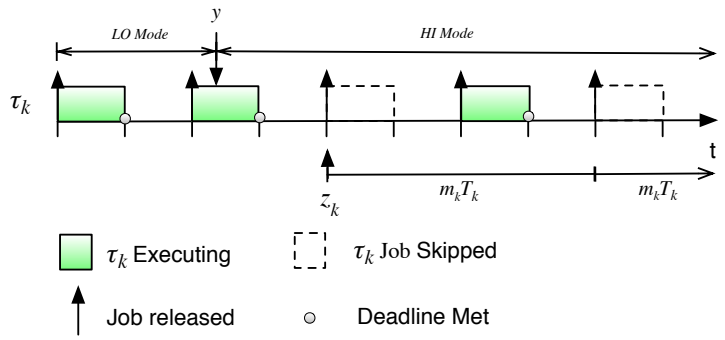

Figure 4.5: Criticality Change of τ_k

1) Schedulability of the Criticality Mode Change

The AMC-max analysis embodied in (4.7), can be modified to incorporate weakly-hard *LO* criticality tasks. The function $M(j, y, t)$ is the same as that used in AMC-max (4.8). This approach removes the pessimism in AMCrtb-WH by assuming HI criticality jobs execute with their C^{LO} values up to the criticality change at time y, at which point active and subsequent HI criticality jobs will execute up to their C^{HI} values. In addition, jobs of each *LO* criticality task τ_k are assumed to exhibit their weakly-hard behaviour, starting consecutive skips at z_k, the first release after the criticality change at y.

For HI criticality tasks, the points at which the triggering of a criticality change y, may affect the response time of a job are bounded by $y = [0, R^{LO})$. If the criticality change were to occur after R_i^{LO} then the job would have already completed its execution. For *LO* criticality tasks being assessed, y should be increased until R^* converges below the current value of y. Once $R^* < y$, increasing the time of the criticality mode change will have no effect on the job and therefore the worst-case response time must have already been obtained.

$$R_i^y = C_i^{L_i} + \sum_{k \in \mathbf{hpLO}(i)} \left(\left\lceil \frac{R_i^y}{T_k} \right\rceil - \sum_{n=s_k}^{m_k} \left\lceil \frac{R_i^y - (m_k - n)T_k - z_k}{m_k T_k} \right\rceil_0 \right) C_k^{LO}$$
$$+ \sum_{j \in \mathbf{hpHI}(i)} \left(M(j, y, R_i^y) C_j^{HI} + \left(\left\lceil \frac{R_i^y}{T_j} \right\rceil - M(j, y, R_i^y) \right) C_j^{LO} \right) \tag{4.14}$$

where $z_k = \left\lceil \frac{y}{T_k} \right\rceil T_k$ and $s_k < m_k$.

The worst-case for the response time for the criticality mode change can be calculated by: $R_i^* = max(R_i^y) \forall y$ where $y \in kT_j \mid \forall j \in hpLO(i) \ \wedge \ y \leq R_i^{LO} \mid \forall k : \mathbb{N}$.

Note that AMC *dominates* AMC-WH since the former effectively skips all jobs of *LO* criticality tasks in *HI* criticality mode. However this means that AMC provides no service for *LO* criticality tasks.

4.2.3 Comparing AMCmax-WH and AMCrtb-WH

This section proves that AMCmax-WH analysis dominates AMCrtb-WH. That is all tasksets that are schedulable under AMCrtb-WH are also schedulable under AMCmax-WH and there exists some taskset that is schedulable under AMCmax-WH, but not under AMCrtb-WH.

Theorem 1. *Any sporadic task system that is schedulable under AMCrtb-WH is also schedulable under AMCmax-WH, hence AMCmax-WH dominates AMCrtb-WH.*

Proof. The analysis for the *LO* and *HI* criticality modes are identical for AMCmax-WH and AMCrtb-WH, therefore only the schedulability tests for the criticality change need be considered.

For *HI* criticality tasks, consider equations (4.12) and (4.14). The equations have three components; (*a*) The WCET of the task, (*b*) the sum of interference from higher priority, *LO* criticality tasks and (*c*) the sum of interference from higher priority, *HI* criticality tasks.

- Component (*a*) of (4.12) and (4.14) are equal.
- Component (*b*) of (4.14) is strictly no larger than component (*b*) of (4.12) as y is upper-bounded by R^{LO} where $R^{LO} \leq R^*$.
- Component (*c*) of (4.14) assumes that higher priority, *HI* criticality tasks execute with their C^{HI} values only after the criticality change. This summation term is therefore maximised when $y = 0$. Substituting this value into (4.14) reduces component (*c*) to that of (4.12). Component (*c*) in (4.14) is therefore upper-bounded by (*c*) in (4.12).

For *LO* criticality tasks, consider equations (4.13) and (4.14), separated into three components; (*a*), (*b*) and (*c*).

- Component (*a*) of (4.13) and (4.14) are equal.
- Component (*b*) of (4.14) is upper-bounded by component (*b*) of (4.13) as $y \leq R^*$ (see subsection 4.2.2).

- Component (c) of (4.13) is identical to that in the HI criticality task analysis discussed above, therefore (c) in (4.14) is upper-bounded by (c) in (4.13).

It has been shown that the result of the schedulability analysis of the criticality mode change in (4.14) can be no greater than (4.12) or (4.13) for $\forall y$.

As the schedulability tests for HI and LO criticality modes are equivalent for AMCmax-WH and AMCrtb-WH it can be deduced that worst-case response times calculated using AMCmax-WH analysis will be no greater than those calculated using AMCrtb-WH for the same taskset, τ. Given Example 6 in section 4.3 which is schedulable under AMCmax-WH but not AMCrtb-WH, it is concluded that AMCmax-WH *strictly dominates* AMCrtb-WH. □

4.2.4 Priority Assignment for AMC-WH

Davis and Burns [39] formalised three Conditions for a schedulability test to be compatible with Audsley's Optimal Priority Assignment (OPA) algorithm [2, 3]:

1. The schedulability of a task τ_k may, according to test $\mathcal{S}$, depend on any independent properties of tasks with priorities higher than τ_k, but not on any properties of those tasks that depend on their relative priority ordering.
2. The schedulability of a task τ_k may, according to test $\mathcal{S}$, depend on any independent properties of tasks with priorities lower than τ_k, but not on any properties of those tasks that depend on their relative priority ordering.
3. When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test $\mathcal{S}$, if it was previously schedulable at the lower priority.

Theorem 2. *AMCrtb-WH and AMCmax-WH schedulability tests comply with the above Conditions [39] and hence Audsley's OPA algorithm can be used to obtain optimal priority ordering.*

Proof. Inspection of equations (4.11) to (4.14) in section 4.2 shows that the schedulability of τ_k under AMC-WH depends only on independent properties of higher priority tasks. As interference is not caused by lower priority tasks, both Conditions (1) and (2) are satisfied. Consider two tasks, τ_j and τ_k with priorities, $P(v)$ and $P(w)$ respectively, where $P(v) > P(w)$. If τ_k is schedulable with priority $P(w)$ under AMC-WH and is swapped with τ_j to acquire priority $P(v)$, interference from higher priority tasks, as calculated by the summation terms, would decrease and so the worst-case response time of τ_k would become less than at $P(w)$, hence τ_k will remain

schedulable. If τ_j is not schedulable at $P(v)$ and is swapped with τ_k to acquire priority $P(w)$, interference from higher priority tasks would increase and so τ_j will remain unschedulable.
This satisfies the two scenarios of Condition (3) and therefore, as with SMC and AMC [14], Audsley's OPA algorithm can be used to find an optimal priority ordering with respect to both AMCrtb-WH and AMCmax-WH schedulability tests. □

4.3 Worked Example

This sections provides a worked example taskset scheduled under the AMC-WH policy to demonstrate the response time analysis developed in section 4.2.

Example 6. *Consider the taskset presented in Table 4.1 for which the OPA algorithm [3] has determined an optimal priority ordering of* $\tau_1 > \tau_2 > \tau 3$.

Task	L_i	C_i^{LO}	C_i^{HI}	D_i	T_i	s_i	m_i
τ_1	HI	1	2	2	4	-	-
τ_2	LO	1	-	4	4	1	2
τ_3	HI	3	3	10	20	-	-

Table 4.1: Example WH Taskset

Schedulability of *LO* Criticality Mode

The schedulability analysis for the LO criticality mode is equivilent for both AMCrtb-WH and AMCmax-WH. Using (4.4)*, the response time values are trivially derived for the taskset. Note the ellipses represent the shortening of the working, i.e these are the converged values found using recurrence relations.*

- $R_1^{LO} = 1$
- $R_2^{LO} = 1 + \left\lceil \frac{R_2^{LO}}{4} \right\rceil * 1 \ \ ... = 2$
- $R_3^{LO} = 3 + \left\lceil \frac{R_2^{LO}}{4} \right\rceil * 1 + \left\lceil \frac{R_2^{LO}}{4} \right\rceil * 1 \ \ ... = 7$

Schedulability *HI* Criticality Mode

The schedulability analysis for the H1 criticality mode is also the same for AMCrtb-WH and AMCmax-WH. Using (4.11) *produces the response time values:*

- $R_1^{HI} = 2$
- $R_2^{HI} = 1 + \left\lceil \frac{R_2^{HI}}{4} \right\rceil * 2 \ \ ... = 3$
- $R_3^{HI} = 3 + \left\lceil \frac{R_3^{HI}}{4} \right\rceil * 2 + \left(\left\lceil \frac{R_3^{HI}}{4} \right\rceil * 1 - \sum_{n=1}^{1} \left\lceil \frac{R_3^{HI} - (2-n)*4}{2*4} \right\rceil_0 \right) * 1 \ \ ... = 8$

Schedulability of Criticality Mode Change for AMCrtb-WH

The criticality mode change analysis is more complex. AMCrtb-WH uses different response time equations depending on if a LO or HI criticality task is being assessed. (4.12) *is used to derive the response times for HI criticality tasks,* τ_1 *and* τ_3.(4.13) *is used for LO criticality task,* τ_2.

- $R_1^* = 2$
- $R_2^* = 1 + \left\lceil \frac{R_2^*}{4} \right\rceil * 2 \ \ ... = 3$
- $R_3^* = 3 + \left\lceil \frac{R_3^*}{4} \right\rceil * 2 + \left(\left\lceil \frac{R_3^*}{4} \right\rceil * 1 - \sum_{n=1}^{2} \left\lceil \frac{R_3^* - (2-n)*4 - x_k}{2*4} \right\rceil_0 \right) * 1 \ \ ... = 11$

  where $x_k = \left\lceil \frac{7}{4} \right\rceil * 4 = 8$

Schedulability of Criticality Mode Change for AMCmax-WH

AMCmax-WH provides tighter analysis for the criticality mode change at the expense of increased complexity. Equation, (4.14)*, is used for all tasks regardless of their assigned criticality level. Note that all relevant values of y have been checked (see subsection 4.2.2), however the working is not listed here for conciseness.*

- $R_1^* = 2$
- $R_2^* = max(R_2^y) \mid \forall y \leq R_2^{LO} \ \ = 3$

  $$R_2^y = 1 + M(1, y, R_2^y) * 2 + \left(\left\lceil \frac{R_2^y}{4} \right\rceil - M(1, y, R_2^y) \right) * 1$$

  where $z_k = \left\lceil \frac{y}{4} \right\rceil * 4$
- $R_3^* = max(R_3^y) \mid \forall y \leq R_3^{LO} \ \ = 8$

$$R_3^y = 3 + \left(\left\lceil \frac{R_3^*}{4} \right\rceil * 1 - \sum_{n=1}^{2} \left\lceil \frac{R_3^* - (2-n)*4 - z_k}{2*4} \right\rceil_0 \right) * 1 + M(1, y, R_3^y) * 2$$
$$+ \left(\left\lceil \frac{R_3^y}{4} \right\rceil - M(1, y, R_3^y) \right) * 1 \qquad where\ z_k = \left\lceil \frac{y}{4} \right\rceil * 4$$

Comparing AMCrtb-WH and AMCmax-WH

Attention is drawn to Table 4.2 which summarises the response times found using the above analysis. Of particular interest is the response time determined by AMCrtb-WH for τ_3 during the criticality mode change (highlighted in bold) which is greater than the assigned deadline for that task. Recalling from section 2.2 that for a taskset to be schedulable under fixed priority the condition $\forall \tau_i \mid R_i \leq D_i$ must hold. It is therefore clear that the example taskset is not schedulable under AMCrtb-WH. Using the less pessimistic analysis of AMCmax-WH however yields response time values for the tasks that are less or equal to their deadlines and the taskset is therefore deemed schedulable.

In subsection 4.2.3 it was shown that AMCmax-WH is upper bounded by AMCrtb-WH; that is response times calculated using AMCmax-WH can be no greater than response times calculated using AMCrtb-WH for the same taskset. Combining this result with this example taskset demonstrates that AMCmax-WH strictly dominates AMCrtb-WH.

Task	L_i	R_i^{LO}	R_i^{HI}	$R_i^*(rtb)$	$R_i^*(max)$	D_i
τ_1	HI	1	2	2	2	2
τ_2	LO	2	3	3	3	4
τ_3	HI	7	8	**11**	8	10

Table 4.2: Summary of Response Times for Example WH Taskset

4.4 Summary

This chapter reviewed existing analysis for mixed criticality systems scheduling based on fixed priority as discussed in chapter 3. A new scheduling policy, AMC-WH, was introduced based on AMC [14] to provide a guaranteed quality of service for LO criticality tasks in HI criticality mode. Response time analysis has been derived for this policy based on AMC-rtb and AMC-max [14] and optimal priority assignment has been proved. The chapter concludes with a worked example.

5 Experimental Evaluation

This chapter reports on an empirical evaluation used to examine the relative performance of the new scheduling policy, AMC-WH, and the associated schedulability tests introduced in section 4.2. A number of experiments were devised in which the new policy is compared to the previous policies reviewed in section 4.1.

5.1 Taskset Generation

A set of uniformly distributed utilisation values were generated using the UUnifast algorithm [20] listed below. The algorithm ensures that n tasks each have a random utilisation value whose sum is no greater than that of the utilisation requested.

Listing 5.1: UUnifast Algorithm

```
SumU := requested utilisation
for i in 0..NumTasks-1 loop
    NextSumU := SumU * Random^(1.0/(NumTasks-i));
    TaskSet(i).Utilisation := SumU - NextSumU;
    SumU := NextSumU;
end loop;
TaskSet(NumTasks).Utilisation := SumU;
```

Periods were then assigned to each task with a log-uniform distribution between 10 and 1000. Log values of T are uniformly distributed to provide a wide range of values of equal probability. Task deadlines were assigned the same values as their periods ($D = T$). C_i^{LO} values were calculated using the utilisation equation $C_i^{LO} = U_i/T_i$ and C_i^{HI} values were assigned by multiplying the C_i^{LO} value by a criticality factor (CF). CP denotes a criticality probability, that is the probability that a particular task will be designated as HI criticality rather than LO criticality.

For the success ratio experiments, 2500 tasksets were generated per utilisation level. For the weighted schedulability [16] tests, a total of 1000 tasksets were generated per utilisation level and value of the varied parameter. By default, each taskset contained 20 tasks with $CP = 0.5$ and $CF = 2.0$.

5.2 Schedulability Tests

UB-H&L - is a composite upper-bound (necessary) schedulability test. Schedulability is assessed for all tasks assuming they execute with their C^{LO} values with DMPO. Separately, all HI criticality tasks are assessed, assuming their C^{HI} values, also using DMPO.
AMC-max - reviewed in subsection 4.1.6, is a tighter analysis than AMC-rtb, taking into account a finite set of points when the criticality change may occur [14].
AMC-rtb - reviewed in subsection 4.1.5, is the response time bound analysis for AMC [14].
SMC - described in subsection 4.1.4. This is SMC with run-time monitoring that deschedules overrunning LO criticality jobs.
SMC-NO - Vestal's original analysis [91] which does not have support for run-time monitoring.
AMCmax-WH - Adaptive Mixed Criticality max - Weakly Hard as described in subsection 4.2.2.
AMCrtb-WH - Adaptive Mixed Criticality response time bound - Weakly Hard as described in subsection 4.2.1.
FPPS - Fixed Priority Preemptive Scheduling. Tasks are in DMPO, ignoring criticality levels. It is noted that this may lead to criticality inversion however for the purpose of these experiments, run-time monitoring is assumed which prevents LO criticality tasks from exceeding their C^{LO} values. Each task assumes its $C_i^{L_i}$ value. Schedulability analysis for FPPS is reviewed in subsection 4.1.1.
CrMPO - Criticality Monotonic Priority Ordering (see subsection 4.1.2) is where tasks are partitioned by criticality level and then DMPO is used within these partitions. This ensures HI criticality tasks are assigned higher priority than any LO criticality task. Response time analysis is then carried out on the tasks using FPPS analysis, where each takes assumes its $C_i^{L_i}$ value. As HI criticality tasks have higher priorities, no run-time monitoring is needed.

5.3 Experiments

Expt.1 (Figure 5.1) - illustrates the schedulability of the different tests at taskset utilisations ranging from 0.05 to 0.95 in steps of 0.05.
Expt.2 (Figure 5.2) - shows the result of altering the Criticality Factor (CF) on schedulability. That is the multiplier between a task's C^{LO} value and its C^{HI} value. This was varied between 1.0 ($C^{HI} = C^{LO}$) and 5.0 in steps of 0.5.
Expt.3 (Figure 5.3) - varies the Criticality Probability (CP) of tasks in a taskset, ranging from 0.05 to 0.95 in steps of 0.05.
Expt.4 (Figure 5.4) - investigates the effect that larger tasksets have on the schedulability tests, ranging from 4 tasks per taskset to 48, in increments of 4.
Expt.5 (Figure 5.5) - is similar to the first experiment, except that the constraint of $D = T$ has been lifted allowing tasks to have deadlines less than their periods. Deadlines were assigned according to uniform random distribution between C_i^{LO} and T_i for LO criticality tasks and C_j^{HI} and T_j for HI criticality tasks.
Expt.6 (Figure 5.6) - varies the number of skips from 0 to 10 in a cycle of fixed length, $m = 10$.
Expt.7 (Figure 5.7) - varies the cycle length, m, from 1 to 10 while keeping the number of skips constant at 1.
Expt.8 (Figure 5.8) - varies the cycle length and skips such that $(s, m) = (m - 1, m)$ for $1 \leq m \leq 10$.

For experiments 1 to 5, the weakly-hard constraints are set at $(s, m) = (1, 2)$ for all LO criticality tasks. This is equivalent to doubling the period while keeping the deadline the same for LO criticality tasks executing in the HI criticality mode. For experiments 2 to 4 and 6 to 8, weighted schedulability [16] is used to flatten the data from 3 vectors to 2. Weighted schedulability is calculated via (5.1) where $S_\phi(\tau, p)$ is a binary test of schedulability of taskset τ with test ϕ and parameter p. Higher utilisation tasksets that are schedulable with test ϕ are more heavily weighted than lower utilisation tasksets.

$$W_\phi(p) = \left(\sum_{\forall \tau} U(\tau) * S_\phi(\tau, p)\right) / \sum_{\forall \tau} U(\tau) \tag{5.1}$$

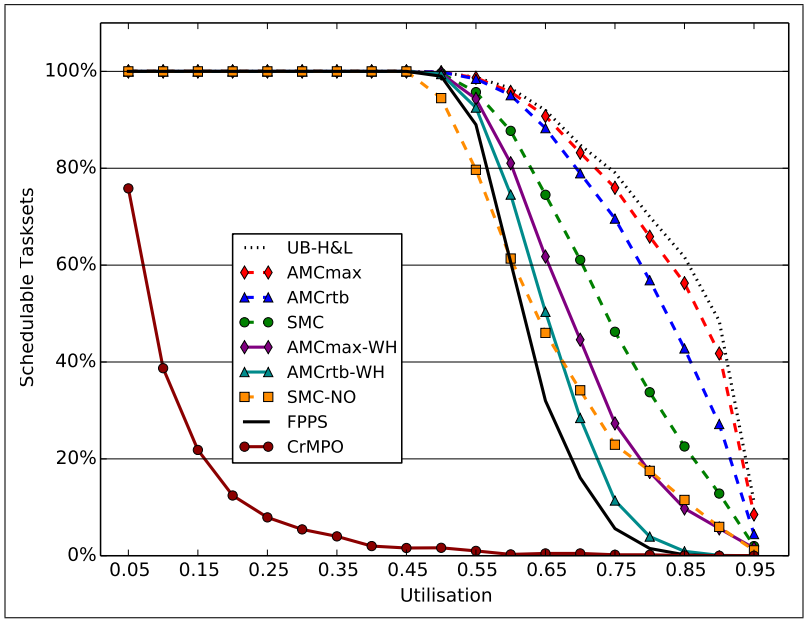

Figure 5.1: **Expt.1** - Percentage of Schedulable Tasksets

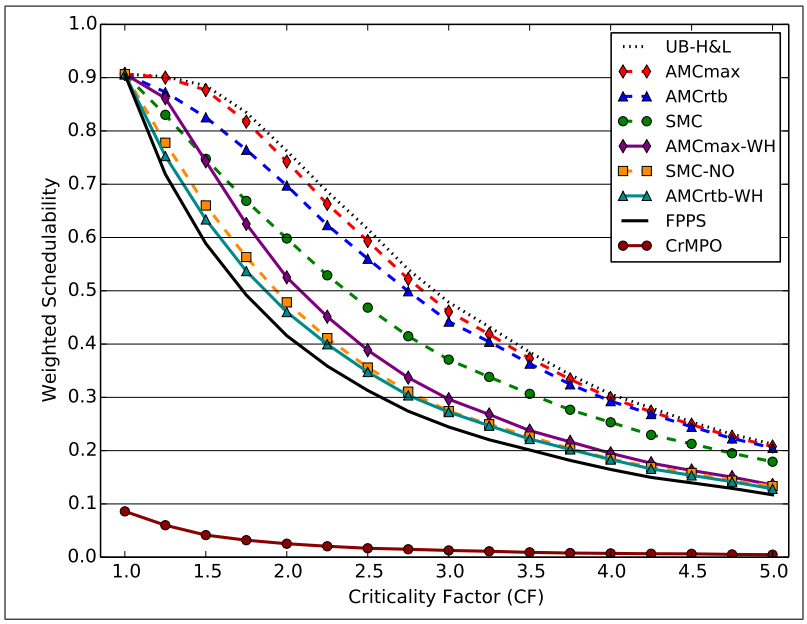

Figure 5.2: **Expt.2** - Varying the Criticality Factor

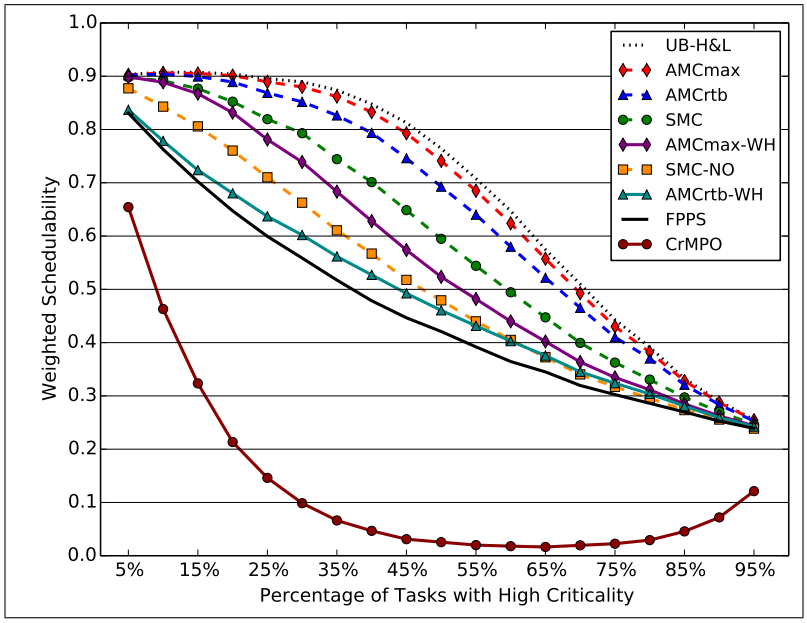

Figure 5.3: **Expt.3** - Varying the Criticality Mix

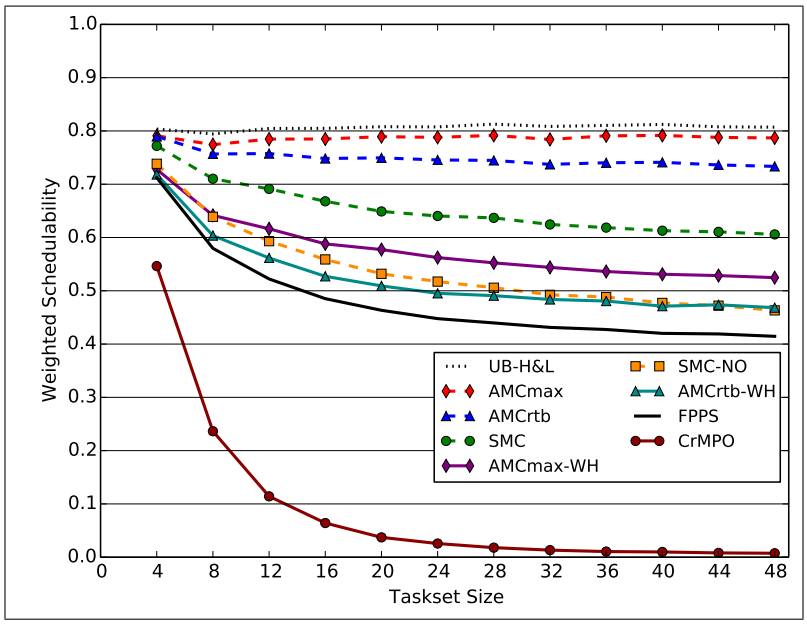

Figure 5.4: **Expt.4** - Varying the Number of Tasks

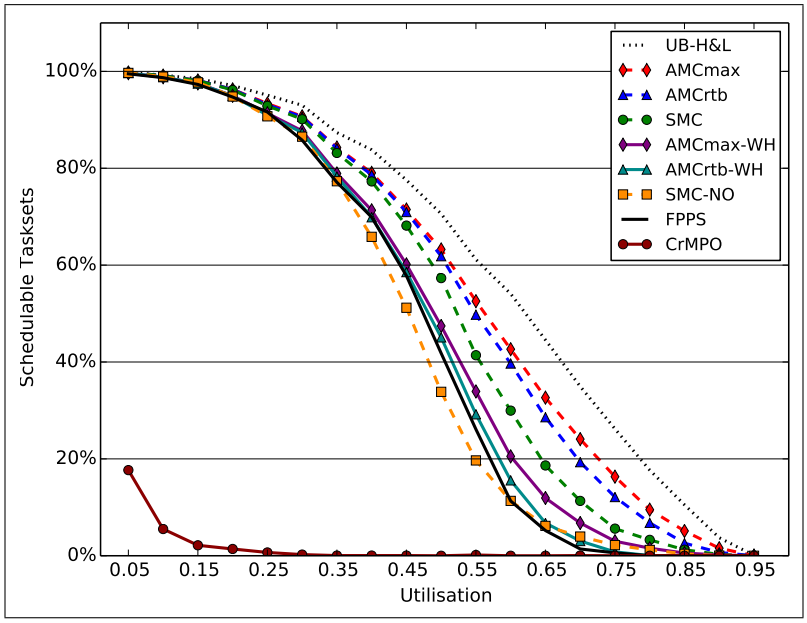

Figure 5.5: **Expt.5** - Percentage of Schedulable Tasksets with $D \leq T$

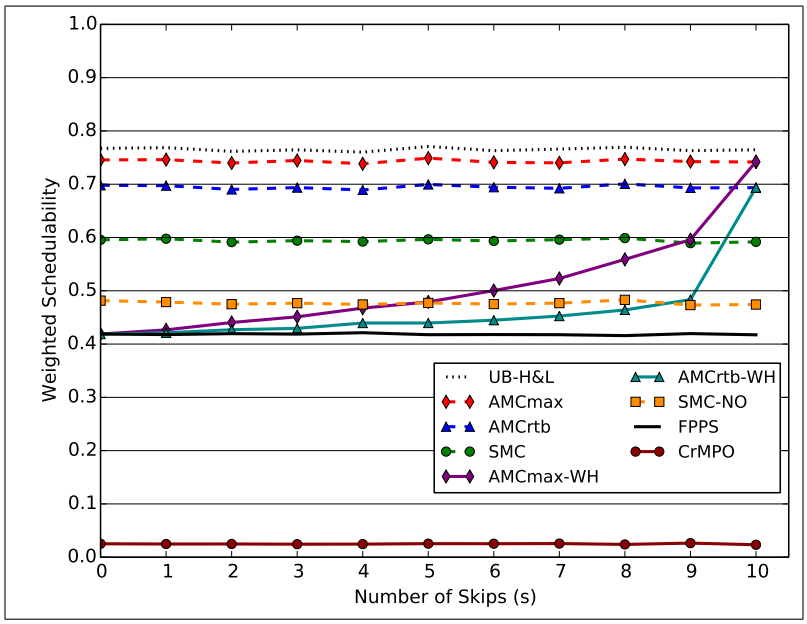

Figure 5.6: **Expt.6** - Varying the Number of Skips where $m = 10$

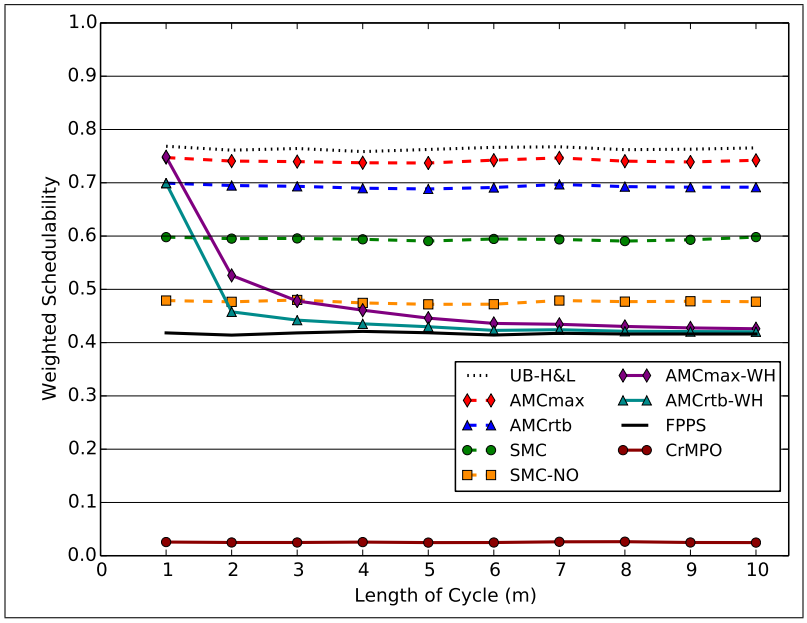

Figure 5.7: **Expt.7** - Varying the Cycle Length where $s = 1$

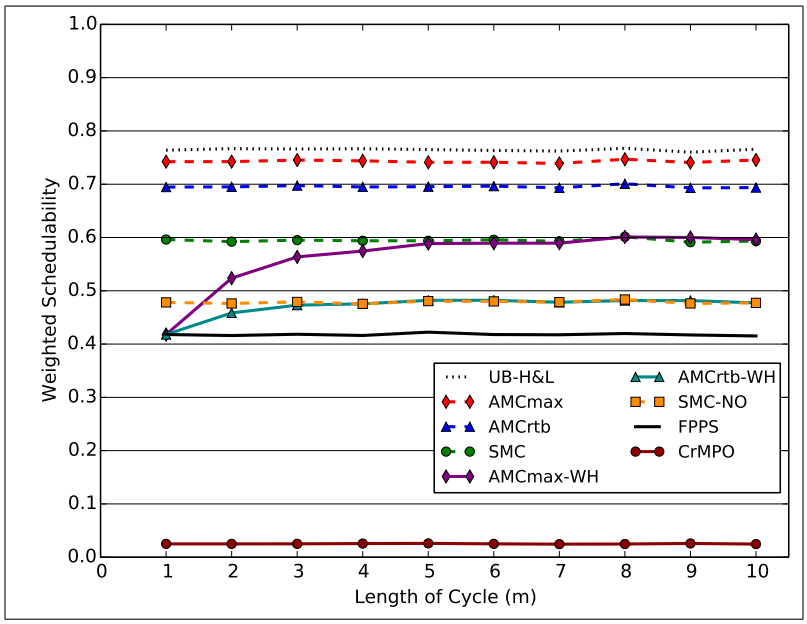

Figure 5.8: **Expt.8** - Varying the Cycle Length where $s = m - 1$

5.4 Discussion of Results

The schedulability tests are grouped into three categories. Solid lines represent tests that guarantee that at least some jobs of LO criticality tasks, assumed to execute up to their C^{LO} estimates, will meet all their deadlines in HI criticality mode. Tests that permit LO criticality tasks to miss their deadlines or deschedule LO criticality tasks in HI criticality mode are represented by dashed lines. The dotted lines on the graphs represent the upper-bounds on schedulability.

A number of points can observed by inspection of the results (Figure 5.1 to Figure 5.8). AMC dominates AMC-WH due to AMC dropping all LO criticality jobs when in HI criticality mode, therefore being schedulable at higher utilisations. The dominance of AMCmax-WH over AMCrtb-WH schedulability tests can be seen across all of the experiments.

Comparing the AMC-WH schedulability tests directly with tests which guarantees LO criticality task deadlines in HI criticality mode, namely CrMPO and FPPS (with run-time monitoring), there is a clear dominance. When AMC-WH is assigned a global value of 100% skips for all LO criticality tasks in HI criticality mode, that is $s = m$, the scheduling behaviour becomes that of AMC. Examining the equations in section 4.2 shows that the schedulability tests for AMCmax-WH and AMCrtb-WH reduce to the equations of AMC-max and AMC-rtb under this condition. This is illustrated in **Expt.6** (Figure 5.6) and **Expt.7** (Figure 5.7). At the opposite extreme, where there are 0% skips for all LO criticality tasks in HI criticality mode, both AMC-WH schedulability tests reduce to the behaviour of FPPS which can be seen in **Expt.6** (Figure 5.6) and **Expt.8** (Figure 5.8). AMC-WH is therefore a compromise between AMC and FPPS, providing scalable performance trade-offs between the quality of service of LO criticality tasks in HI criticality mode and the schedulability of the HI criticality mode.

Expt.8 (Figure 5.8) illustrates a behaviour similar to the work of Yip *et al.* [96] in terms of AMC-WH, where a task's period is extended while its relative deadline remains constant, resulting in lower utilisation.

5.5 Additional Investigation

Declaration: The following paragraph was written by Dr R. I. Davis:
The LO criticality budget C^{LO} set by the system designer for each HI criticality task is typically obtained via a process which involves rigorous testing and measurement on the target hardware. This process produces a high watermark execution time, which

may be further inflated by some engineering margin to obtain the LO criticality budget, C^{LO}. Hence C^{LO} is in practice highly unlikely to be exceeded at runtime. Nevertheless, this is not enough to guarantee to a sufficiently high level of assurance that each HI criticality task will not exceed its deadline. Hence we also have the HI criticality budget C^{HI} determined via an even more stringent and conservative process, and thus the mixed criticality scheduling model. In this section, we investigate the impact on schedulability of the following assumption (A1): Only a single HI criticality task can exhibit HI criticality behaviour in any given busy period. We note that to be justified such an assumption needs a careful argument to be made about the independence or limited dependencies between task execution times, and about the probabilities that each job of a task can exceed its C^{LO} budget. This argument would then be used to show that the probability of jobs of two or more different tasks exhibiting HI criticality behaviour within a short time window (equating to a busy period) is so vanishingly small that it can be safely ignored. Here we *do not* attempt to make this argument, which could potentially fill a whole paper with interesting discussion. Rather we simply seek to characterise the improvements in schedulability that can be obtained *if* the assumption (A1) can be relied upon to hold.

Under the assumptions described above, **Expt.1-8** were repeated with the same parameters outlined in section 5.3. To assess the worst-case response time, multiple schedulability tests were required, with each HI criticality task assumed to be the one task that exhibits HI criticality behaviour.

Enforcing this constraint has a huge effect on the schedulability of tasksets, as can be seen in **Expt.1b** (Figure 5.9). Here UB-L&H-1 is used to illustrate the upper-bound on schedulability for these policies. Here, SMC-NO-1 performs considerably worse when compared to the relative performance spread of the previous experiments. This is due to the static nature of SMC-NO-1 which requires higher priority, LO criticality tasks to be verified using their C^{HI} values reducing the advantages of assuming constrained HI criticality task behaviour.

Expt.2b (Figure 5.10) illustrates the advantage of assuming only one HI criticality task executes with its C^{HI} value in HI criticality mode. In **Expt.2** (Figure 5.2), increasing the CF significantly reduces the schedulability whereas **Expt.2b** has an almost linear degradation in schedulability. This is due to the freed up utilisation in HI criticality mode which accommodates the increased C^{HI} value of the one HI criticality task which exhibits HI criticality behaviour. **Expt.3b** (Figure 5.11), varying the criticality mix, shows similar gains in schedulability. Increasing the mix has a

limited effect on the schedulability of the tasksets since it is assumed that there will be at most one HI criticality task exhibiting HI criticality behaviour at any given time.

Expt.4b (Figure 5.12) in contrast to **Expt.4** (Figure 5.4) shows that schedulability increases with the size of the taskset. This is due to the ratio of tasks executing in HI criticality mode (1) and LO criticality mode ($N-1$) becoming smaller resulting in tasksets being more easily schedulable in HI criticality mode. As the number of tasks in a taskset increases, the system tends to the schedulability of a single criticality system. The exception to this observation is CrMPO-1 and SMC-NO-1. SMC-NO-1 as discussed previously requires higher priority, LO criticality tasks to verified up to the criticality level of the highest criticality task to which they may cause interference. Despite only one HI criticality task being permitted to execute with its C^{HI} value at any given time, all HI criticality tasks are ordered in the same way as standard CrMPO therefore CrMPO-1 has a suboptimal priority ordering.

Expt.5b (Figure 5.13) showed only modest gains in schedulability compared to **Expt.5** (Figure 5.5). With constrained deadlines, the LO criticality mode and criticality mode change are more difficult for the polices to schedule. Therefore increasing the schedulability of the HI criticality mode under our assumptions has a limited effect on the schedulability of the entire taskset. **Expt.6b**, **7b** and **8b** (Figure 5.14 to Figure 5.16) all demonstrated large gains in schedulability, consistent with reduced utilisation of the system in HI criticality mode.

The main observation from these results is that the schedulability of a mixed criticality taskset can be significantly increased under the assumption of restricted HI criticality task behaviour discussed above. The potentially large increase in schedulability warrants further research into the arguments required to ensure that the assumption (A1) holds.

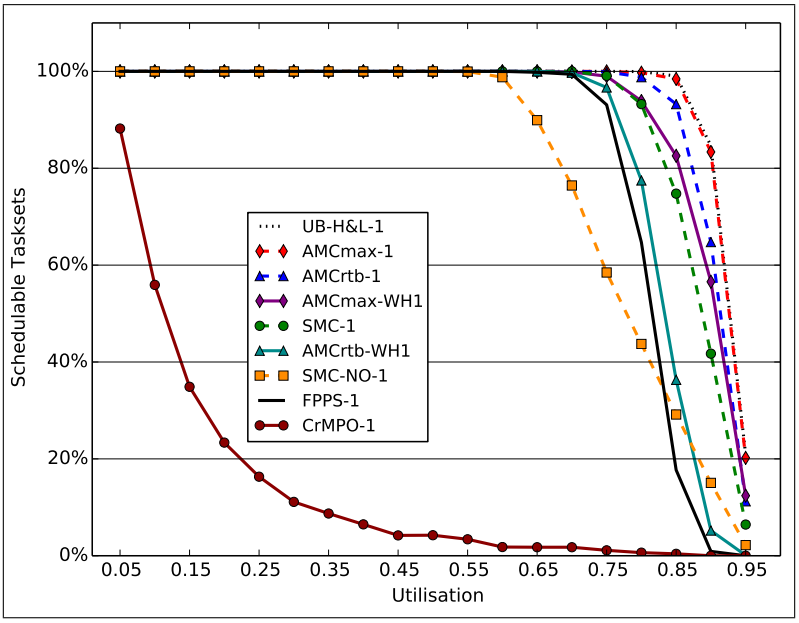

Figure 5.9: **Expt.1b** - Percentage of Schedulable Tasksets (1-HC)

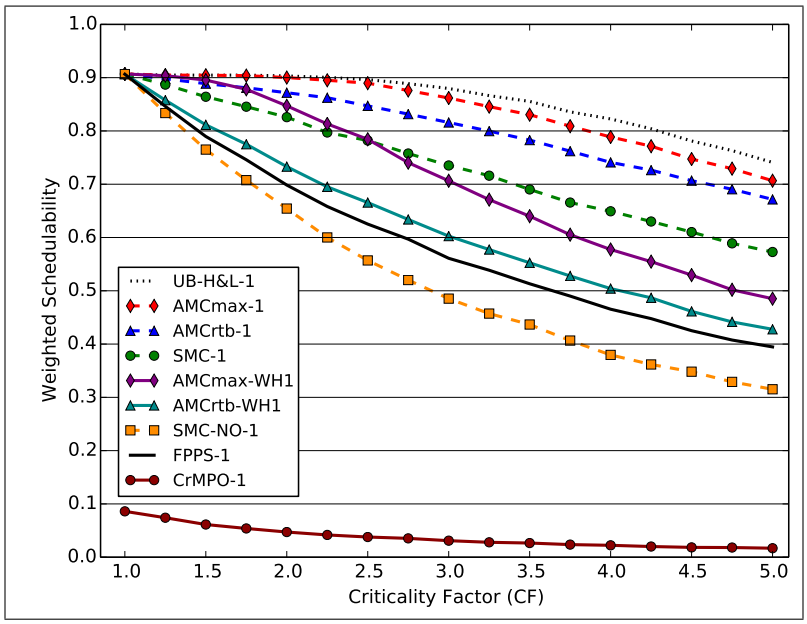

Figure 5.10: **Expt.2b** - Varying the Criticality Factor (1-HC)

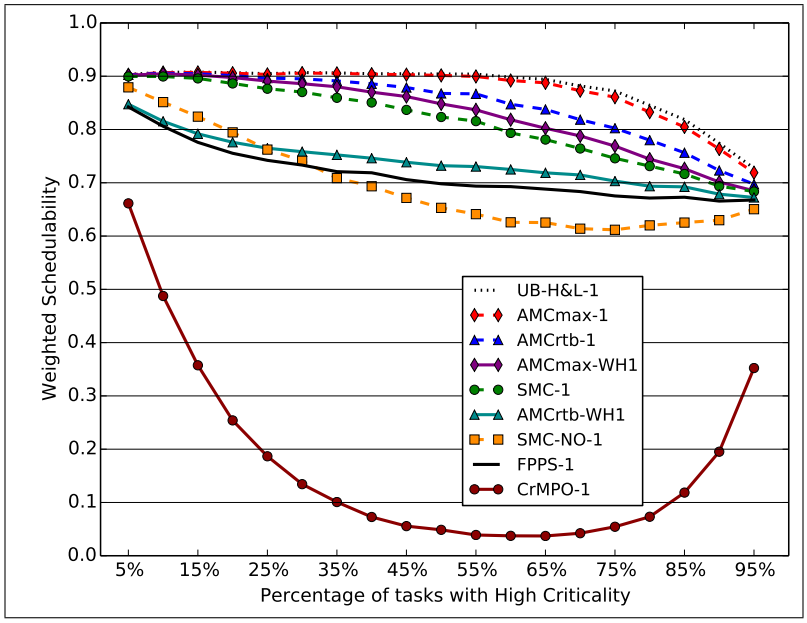

Figure 5.11: **Expt.3b** - Varying the Criticality Mix (1-HC)

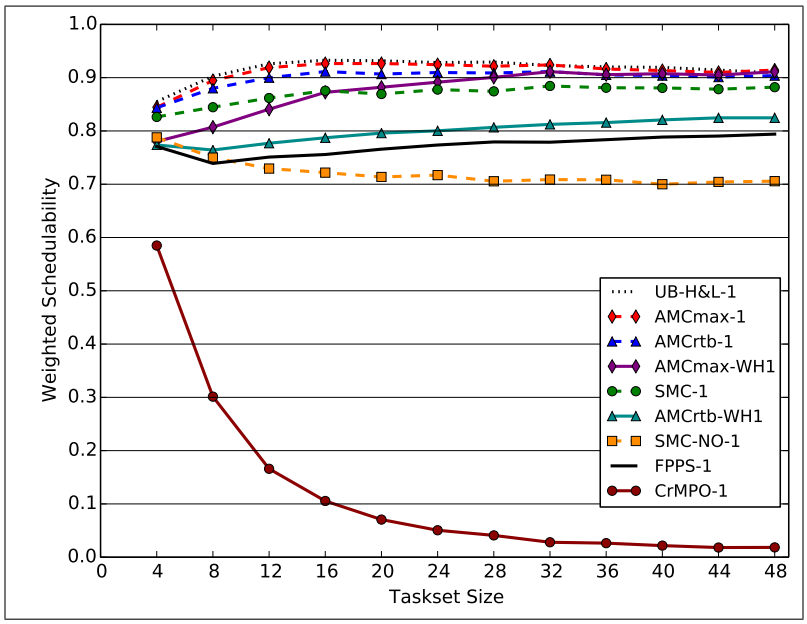

Figure 5.12: **Expt.4b** - Varying the Number of Tasks (1-HC)

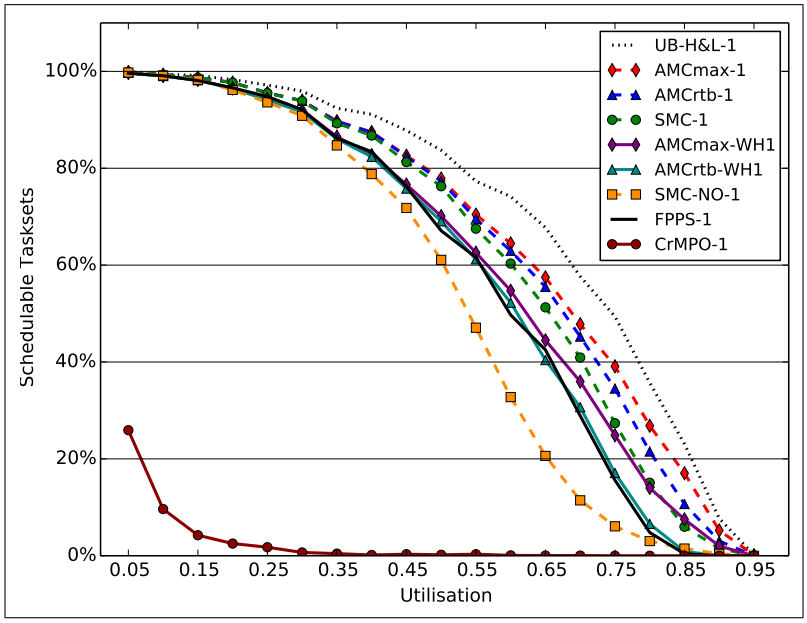

Figure 5.13: **Expt.5b** - Percentage of Schedulable Tasksets with $D \leq T$ (1-HC)

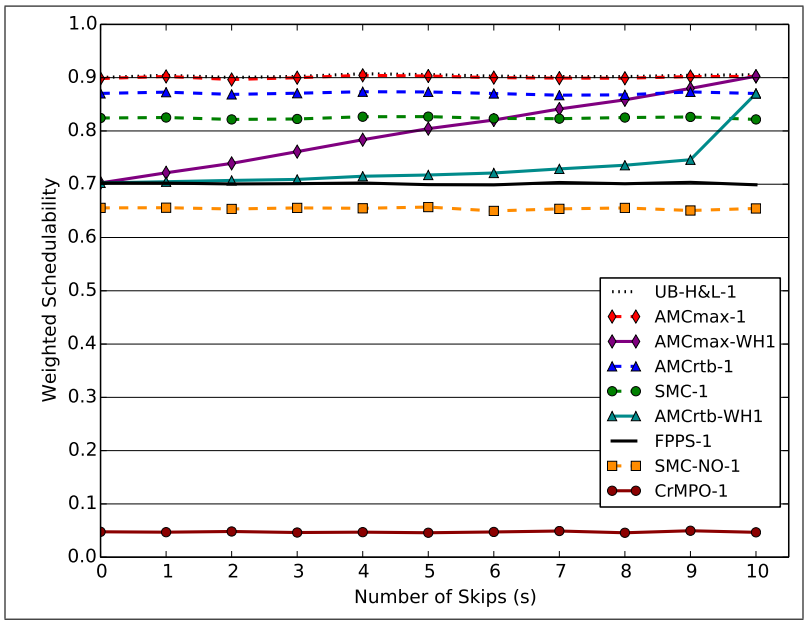

Figure 5.14: **Expt.6b** - Varying the Number of Skips where $m = 10$ (1-HC)

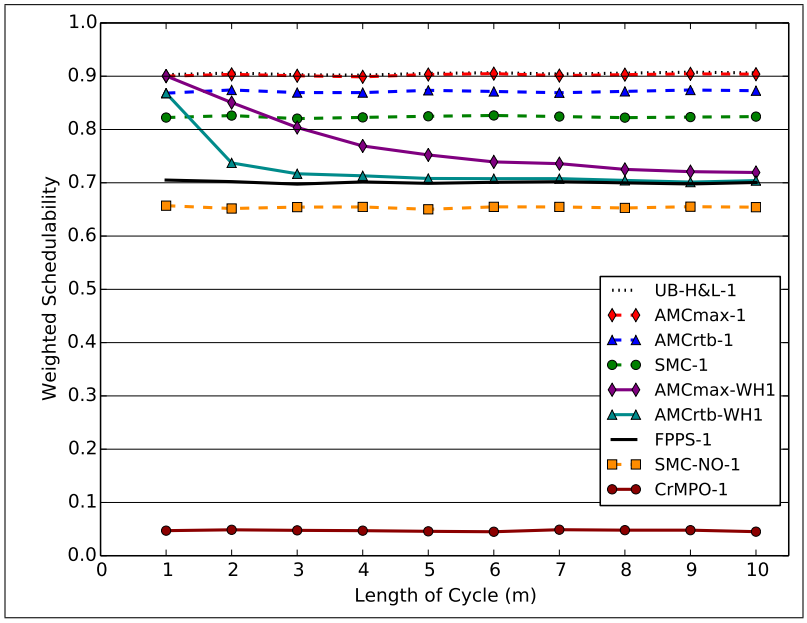

Figure 5.15: **Expt.7b** - Varying the Cycle Length where $s = 1$ (1-HC)

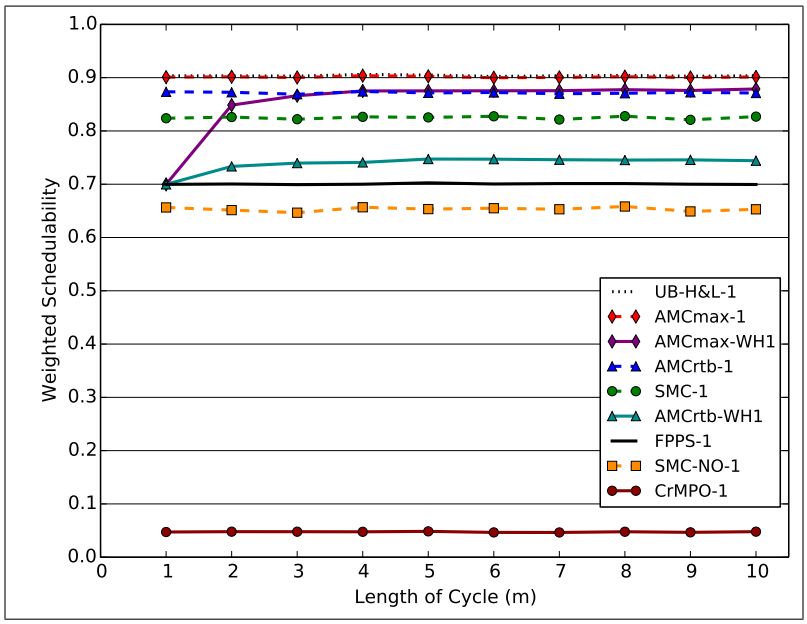

Figure 5.16: **Expt.8b** - Varying the Cycle Length where $s = m - 1$ (1-HC)

5.6 Summary

This chapter reviews an extensive empirical investigation into the effectiveness of AMC-WH (introduced in section 4.2) against existing scheduling policies for MCS. By utilising weighted schedulability [16] it has been possible to vary multiple parameters such as the size of a taskset or the criticality mix to assess the effect on schedulability. AMC-WH was shown to perform well when compared to other policies that guarantee LO criticality behaviour in the HI criticality mode. While this is partly due to the reduced quality of service offered, AMC-WH was shown to offer scalable performance between FPPS and AMC.

An additional investigation was carried out under the strict assumption that only one HI criticality task may execute with its full C^{HI} value at any one time. This was shown to significantly increase the schedulability of tasksets.

6 Conclusions

This work has reviewed the principles of schedulability analysis for real-time systems, examining the various approaches to ensure the predictability of such systems. It then continued with a survey of the active research areas of mixed criticality systems on uni-processor platforms. A number of technical issues impeding the adoption of mixed criticality systems have been discussed with particular focus on the issue of providing a level of *LO* criticality behaviour in the event of a criticality mode change.

The main contribution of our research is the scheduling policy, AMC-WH, introduced in section 4.2. This policy can be used to ensure a minimum Quality of Service (QoS) for *LO* criticality tasks in the event of a criticality mode change, an imperative issue if such a policy is to be deployed on real systems. Empirical evaluations demonstrated that AMC-WH performs favourably with respect to existing policies, exhibiting a reasonable reduction in schedulability to accommodate the continued execution of *LO* criticality tasks without compromising the assurance of *HI* criticality tasks. The approach provided by AMC-WH offers the flexibility for per-task constraints, allowing a system designer to dictate the level of QoS for a particular component in the event of entering the *HI* criticality mode. This opens up the possibility of combining AMC-WH with the notion of importance developed by Fleming and Burns [50].

Areas of possible future work include generalising AMC-WH to n-criticality levels, as has been done with AMC-rtb and AMC-max [49]. Generalise the model to allow all tasks to exhibit weakly-hard behaviour in any criticality mode, where each task is assigned a set of constraints (one per criticality level). This will offer additional flexibility to the system designer when deciding how the system should degrade after a criticality mode change. Another interesting avenue of research is to explore the use of sensitivity analysis to derive the weakly-hard parameters needed for schedulability under AMC-WH. This approach would allow the highest possible quality of service for *LO* criticality tasks. It is also advantageous to investigate the integration of methods for the rapid recovery to *LO* criticality mode [17] with the AMC-WH policy.

A Investigation: Sub-optimality of Fixed Priority Scheduling

In 1995 Kalyanasundaram and Pruhs [63] investigated the effect of speeding up a processor on the schedulability of tasksets using different scheduling algorithms. The ratio of this processor speedup, that is the ratio of the processor speeds at which a taskset becomes *just* schedulable under each scheduling algorithm is referred to as the speedup factor. This can be consider a measure of relative effectiveness of a scheduling algorithm.

To define a speedup factor in more concrete terms, consider $f^{Opt}(\tau)$ as the lowest processor speed such that taskset, τ, is *just* schedulable under an optimal scheduling algorithm, Opt. Consider $f^{A}(\tau)$ as the lowest processor speed such that taskset, τ, is *just* schedulable under a suboptimal scheduling algorithm A. The speedup factor f^{A} is expressed [35, 63] as:

$$f^{A} = \max_{\forall \tau} \left(\frac{f^{A}(\tau)}{f^{Opt}(\tau)} \right) \tag{.1}$$

A summary of the upper bounds and lower bounds derived for the sub-optimality of FP vs EDF as of 2014 are shown in Table .1 [40, 42, 73].

	FP-P vs EDF-P	
Task Constraint	Lower Bound	Upper Bound
Implicit-deadline	$1/ln(2) \approx$ **1.44269**	
Constrained-deadline	$1/\Omega \approx$ **1.76322**	
Arbitrary-deadline	$1/\Omega \approx$ **1.76322**	**2**
	FP-NP vs EDF-NP	
Task Constraint	Lower Bound	Upper Bound
Implicit-deadline	$1/\Omega \approx$ **1.76322**	**2**
Constrained-deadline	$1/\Omega \approx$ **1.76322**	**2**
Arbitrary-deadline	$1/\Omega \approx$ **1.76322**	**2**

Table .1: Summary of Theoretical Upper Bounds on Speedup Factors

Exact bounds are known for implicit and constrained deadlines for the preemptive case. Exact bounds for implicit and constrained deadlines for the non-preemptive case were later given by Bruggen *et al.* [92] as $1/\Omega$.

The exact bounds for arbitrary deadlines for both the preemptive and non-preemptive cases are not known [35, 37]. This investigation aimed to discover, if possible, improved bounds for these cases by the use of a genetic algorithm.

Taskset Generation

A suitable number of tasksets need to be generated to provide an adequate search-space for discovering individuals with large speedup factors. For this requirement, Enrico Bini's UUnifast algorithm [20] is used to create a random uniform set of task utilisation values totalling that of a requested utilisation.

Only tasksets with utilisation greater than 1.0 were created. The reason for this requirement is that should a utilisation of a taskset be less than one, then the processor speed required to just schedule the taskset under EDF-P would be also less than 1.0 (since EDF is always schedulable when $U \leq 1$ [73]). This may lead to granularity problems should task attributes have to be reduced beyond their base values. With a utilisation greater than 1.0, the binary search will move in a positive direction for both FP and EDF.

A period value, T_i, for each task, τ_i, is selected from a log uniform distribution in the range [1,1000]. Since U_i has already been assigned by the UUnifast algorithm and $U_i = C_i/T_i$, the computation time can be calculated by $C_i = U_i * T_i$. Deadlines for tasks are assigned a random log uniform value between C_i and $10T_i$.

To reduce issues with granularity, 64-bit integer are used rather than long floats for task attributes. This means that to have reasonable level of accuracy during division operations used in the schedulability analysis, at least 5 decimal places should be simulated by factoring up the integer values of C_i, T_i and D_i by 10^5.

Speedup Calculation

The speedup factor for Fixed Priority Preemptive (FP-P) scheduling vs Earliest Deadline First Preemptive (EDF-P) scheduling can be expressed as

$$f^{FP-P} = \max_{\forall \tau} \left(\frac{f^{FP-P}(\tau)}{f^{EDF-P}(\tau)} \right) \tag{.2}$$

The speedup factor for Fixed Priority Non-Preemptive (FP-NP) scheduling vs Earliest Deadline First Non-Preemptive (EDF-NP) scheduling can be expressed as

$$f^{FP-NP} = \max_{\forall \tau} \left(\frac{f^{FP-NP}(\tau)}{f^{EDF-NP}(\tau)} \right) \tag{.3}$$

To further reduce granularity problems, rather than scaling C_i in tasksets (which is the smallest task attribute), D_i and T_i were instead scaled to simulate the change in processor speed.

The process of the binary search for the speedup factor of a taskset, τ, is as follows:

1. Populate a taskset, τ, consisting of n tasks using a requested utilisation > 1.0.
2. Use binary search to adjust the speed of the processor by applying a multiplier (*MedMulti*) to D_i and T_i for each task τ_i in τ.
3. Once the multiplier has been applied, τ is tested for schedulability under f^A.
4. Should τ be unschedulable, a higher multiplier is applied, such that:

   $LowMulti := MedMulti$
   $MedMulti := LowMulti + (HiMulti - LowMulti)/2$
5. Else, if τ is schedulable, a lower multiplier is applied such that:

   $HiMulti := MedMulti$
   $MedMulti := LowMulti + (HiMulti - LowMulti)/2$
6. When *MedMulti* is a distance of 0.01% from *HiMulti* and *LowMulti*, the search is stopped and *LowMulti* is taken as the f^A speedup.
7. The process is repeated for the optimal scheduling algorithm f^{Opt} taking *HiMulti* as the speedup value.
8. The speedup factor is calculated using the equations above.

Taking a lower value from the suboptimal scheduling algorithm and a higher value from the optimal algorithm when calculating the speedup factor avoids the result being optimistic.

An initial value of 1.0 is used for *LowMulti* as this would be the minimum multiplier required. The reasoning behind this is that the initial utilisation of a taskset will be greater than 1.0, requiring the processor to be speeded up for a taskset to be feasible, even under an optimal scheduling algorithm. If mutation causes a taskset to require a processor multiplier less than 1.0, the resultant speedup factor would be so low that it would be of no interest and would be discarded by the

evolutionary algorithm selection process (see section 6). After 3 weeks of tuning parameters to increase the performance of the genetic algorithm, the initial value of $HiMulti$ was selected to be 500000.0. This was determined to be the maximum value that would not cause floating-point errors in the implementation. While it can be argued that this artificially restricted the search space, during the entire investigation no randomly generated taskset required a speedup multiplier approaching this limit. The initial value of $MedMulti$ is simply $(HiMulti)/2$.

Schedulability Analysis

For simplicity, the analysis implemented does not accommodate release jitter and for the preemptive cases, does not account for blocking.

FP-P

A version of Equation 2.5 omitting jitter and blocking has been implemented to test schedulability under FP-P for tasksets with arbitrary deadlines. As a preliminary test, the utilisation of a taskset is calculated. It is known that a taskset with $U > 1.0$ will not be feasible and so can be immediately returned as unschedulable. However, the actual condition checked is that $U \leq 0.99999$. This is due to the limitations in calculation of L_a in the EDF analysis. It should be noted that this imposed limitation in the implementation can result in a speedup factor value error of 0.001%. This small error will only affect tasksets with speedup factors close to 1.0 which, in most cases, will be discarded by the evolutionary algorithm.

$$w_i^{n+1}(q) = (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n(q)}{T_j} \right\rceil C_j \tag{.4}$$

$$r_i(q) = w_i^n(q) - qT_i \qquad\qquad R_i = max_{q=0,1,2...} R_i(q) \tag{.5}$$

As with the jitter equation, increasing of q can stop once $r_i(q) \leq T_i$. Each R_i needs to be checked for the condition $R_i \leq D_i$ to ensure that a taskset is schedulable.

FP-NP

Similar to equations Equation 2.8-Equation 2.11, albeit with jitter removed, the analysis for FP-NP is calculated using the equations below.

$$w_i^{n+1}(q) = qC_i + \sum_{j \in hp(i)} \left(1 + \left\lfloor \frac{w_i^n(q)}{T_j} \right\rfloor \right) C_j + B_i^{NP} \tag{.6}$$

Where

$$B_i^{NP} = \max_{\forall k \in lp(i)} (C_k - \Delta) \tag{.7}$$

All releases of τ_i in the level-i busy period need to be assessed to find the worst case response time.

$$R_i = max_{q=0,\ldots,Q-1}(W_i(q) - qT_i + C_i) \tag{.8}$$

The last q that needs to be assessed is given by:

$$Q = \left\lceil \frac{L_i}{T_j} \right\rceil \tag{.9}$$

If $\forall i \mid R_i \leq D_i$, the taskset is schedulable under FP-NP.

EDF-P

The processor demand function used is that of Equation 2.14 while Equation 2.16-Equation 2.18 have been modified to remove jitter and blocking. The maximum feasible utilisation has been reduced from 1.0 to 0.99999. This is to limit the maximum length of the level-i busy period and ensure that schedulability tests can be completed in reasonable time. It is noted that this will result in the schedulability test being slightly pessimistic.

The result of this pessimism is that discovery of the exact highest possible speedup factor will not be possible, however it does ensure that optimistic speed up factors that exceed actual upper bounds are not erroneously returned.

$$L_a = max\left\{ (D_1 - T_1), \ldots, (D_n - T_n), \frac{\sum_{i=1}^{n}(T_i - D_i)(C_i/T_i)}{1 - U} \right\} \tag{.10}$$

$$w^0 = \sum_{i=1}^{N} C_i \qquad w^{j+1} = \sum_{i=1}^{N} \left\lceil \frac{w^j}{T_i} \right\rceil C_i \quad when \quad w^j = w^{j+1}, \quad L_b = w^{j+1} \tag{.11}$$

$$L = \begin{cases} min(L_a, L_b) & U < 0.99999 \\ L_b & U = 0.99999 \end{cases} \tag{.12}$$

The QPA algorithm that has been implemented is simplified and does not support analysis with jitter or blocking.

Listing 1: QPA Algorithm without Jitter or Blocking

```
t := max{d_i | d_i < L}
while (h(t) ≤ t and h_j(t) > D_min) {
    if(h(t) < t)
        {t := h(t)}
    else
        {t = max{ d_i | d_i < t}}
}
if (h(t) ≤ D_min) { schedulable }
else { not schedulable }
```

EDF-NP

EDF-NP analysis is similar to Equation 2.22 and Equation 2.23 in subsection 2.3.5, the only difference being the removal of jitter.

$$\forall t \leq L, \ \sum_{i=1}^{N} max\left(0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor\right) C_i + b^{np}(t) \leq t \tag{.13}$$

Where

$$b^{np}(t) = \begin{cases} 0 & \nexists i \ : D_i > t \\ \max\limits_{D_i > t}\{C_i - 1\} & others \end{cases} \tag{.14}$$

Listing 2: QPA Algorithm without Jitter

```
t := max{d_i | d_i < L}
while (h(t) + b(t) ≤ t and h(t) + b(t) > D_min) {
    if(h(t) + b(t) < t)
        {t := h(t) + b(t)}
    else
        {t = max{ d_i | d_i < t}} }
if (h(t) + b(t) ≤ D_min) { schedulable }
else { not schedulable }
```

Evolutionary Algorithm

Although searching a random population of tasksets may eventually yield a particular taskset with a high speedup factor, it is more effective to use an evolutionary algorithm to evolve tasksets that characteristically have high speedup factors.

A genetic algorithm (GA) was implemented to perform this task. A population of N tasksets each with n tasks and an initial taskset utilisation of U, are created using the process outlined above. These tasksets are subjected to the speedup calculation process. Speedup factors are assigned to each taskset to be used as the fitness values in the genetic algorithm.

The population then goes under a tournament selection process[1] and are crossed over by splicing tasksets to produce two offspring. These crossed over individuals are subjected to mutation. The process is repeated until there are $2 * N$ individual tasksets. A final tournament selection takes place to decide which tasksets will survive. This is performed on the newly created child population and the parent population until there are N tasksets to construct the next generation. The entire process is repeated, tasksets with higher speedup factors being allowed to survive more often than tasksets with lower speedup factors. An outline of the GA is illustrated in Figure .1.

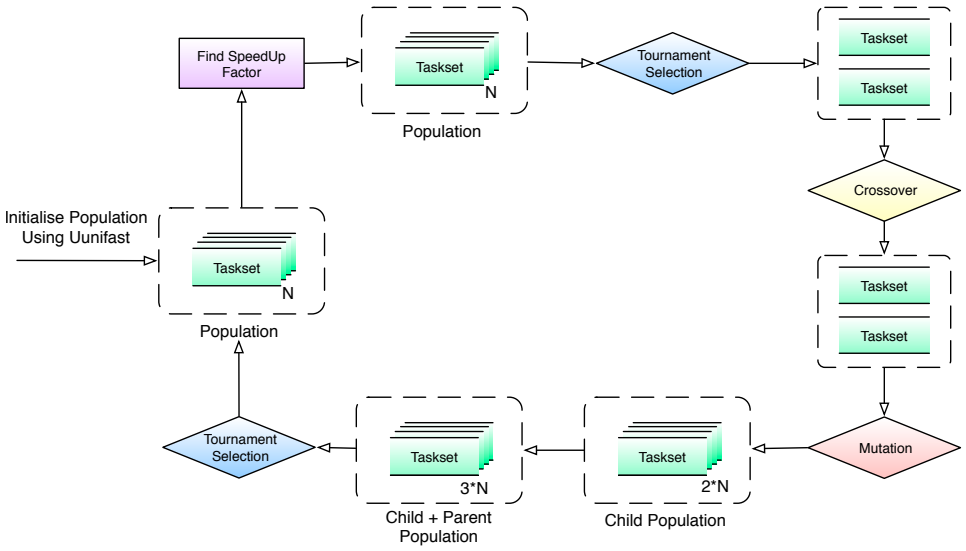

Figure .1: Flow Diagram of Genetic Algorithm

[1]Probability selection based on fitness values.

Crossover

The crossover used in this genetic algorithm is a simple one-point crossover, that is two tasksets of equal size are taken and a random point is chosen along their length. The tasksets are then split and recombined in a head and tail fashion. There is a probability that crossover may not occur at all. If crossover does not occur, the two tasksets are returned in their original state and passed to the mutation function.

Mutation

The mutation function is designed to explore the whole search-space. The three attributes of a task, τ_i, have an equal chance of being mutated, although only one is permitted to be altered per mutation. There is a global mutation probability that dictates if mutation should occur. If this should evaluate to be true, C_i, D_i or T_i are randomly selected for mutation. A value in the random range of (0,20]% is added or subtracted from the selected attribute.

Tournament

Tournament selection is a probabilistic selection of individuals from a population where individuals with higher fitness values are more likely to be selected than those with lower fitness values. A number of tasksets are randomly selected from the existing population to form a tournament population. Tasksets are then selected for survival based on their fitness values (speedup factors).

The size of a tournament population is given by the tournament size attribute. The larger the size, the greater the selection pressure. If the tournament size equals the initial population size then it is the equivalent to elitism, where only the individuals with the highest fitness values are selected.

Parameters

The parameters used in the genetic algorithm are summarised in Table .2. These parameters were tuned over a period of 3 weeks to produce the highest speedup factors. Tournament size in particular had a significant effect on the maximum speedup factors yielded. A lower tournament size has a lower selection pressure allowing a variety of tasksets, including those with lower speedup factors to survive. Having a genetic variety in a population can reduce the chance of the GA becoming stuck in a local optima.

Representation	Array of Tasksets
Crossover Type	1-Point Crossover
Crossover Probability	0.5
Mutation	$\pm$ (0-20]% Deadline, CompTime or Period
Mutation Probability	0.6
Parent Selection	Tournament
Survival Selection	Tournament on Old and New Population
Population Size	100,000
Tournament Size	50
Generations	200

Table .2: Genetic Algorithm Parameters

Results

FP-P vs EDF-P, $D \sim T$

The genetic algorithm was able to find a taskset that has a higher speedup factor than the lower bound of ≈ 1.76322 given in [37] (see Table .1). This taskset has a speedup factor of 1.80454 which makes it a potentially new best case and improved lower bound for the the preemptive scheduling case with arbitrary deadlines. Table .3 gives the taskset attributes.

ID	TimePeriod	Deadline	CompTime
1	134359	720446	31653
2	365919	739991	84855
3	162077	758564	30229
4	88421	755213	15537
5	11342945	822606	852448
6	336043	668423	123684
7	146577	89018	283
8	586012	633882	215465
SpeedUp	**1.80454**		
U.Bound	$[\approx 1.76322, 2]$		

Table .3: FP-P vs EDF-P for $D \sim T$

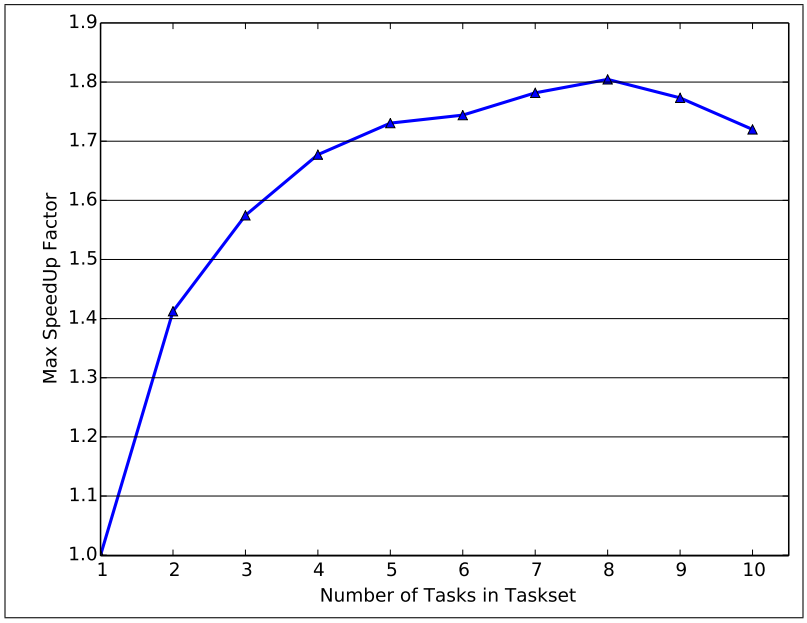

Figure .2: Speedup Factors for FP-P vs EDF-P, $D \sim T$

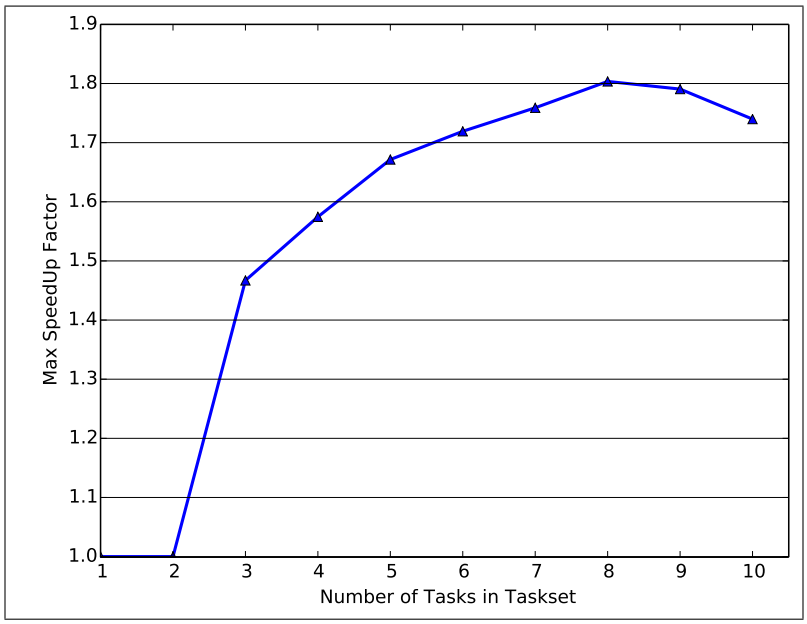

Figure .3: Speedup Factors for FP-NP vs EDF-NP, $D \sim T$

FP-NP vs EDF-NP, $D \sim T$

Similar to the preemptive case for arbitrary deadlines, the genetic algorithm produced a taskset that exceeds the lower bound (≈ 1.76322) given in [35], providing a potentially new best case of 1.80352. The taskset attributes are listed in Table .4.

ID	TimePeriod	Deadline	CompTime
1	1889920	95475	17502
2	70560	86069	18553
3	27012	88689	8714
4	36818	89393	11851
5	3058493	643262	137633
6	16976	87339	6749
7	25997	81015	12331
8	13758	84752	6234
SpeedUp	**1.80352**		
U.Bound	$[\approx 1.76322, 2]$		

Table .4: FP-NP vs EDF-NP for $D \sim T$

FP-P vs FP-NP

An investigation into FP-P vs FP-NP was also conducted, for which the speedup factor is expressed as:

$$f^{FP-PNP} = \max_{\forall \tau} \left(\frac{f^{FP-NP}(\tau)}{f^{FP-P}(\tau)} \right) \tag{.15}$$

Taskset generation and binary search were performed according to the process described above, using the same parameters. The results of these experiments are illustrated in Figure .4. At the time of the investigation tight bounds were not known, however Davis *et al.* [34, 38] have since formalised the upper and lower bounds for these cases:

	FP-P vs FP-NP	
Task Constraint	Lower Bound	Upper Bound
Implicit-deadline [73]	*unkown*	$1/ln(2) \approx$ **1.44269**
Constrained-deadline [34]	$\sqrt{2} \approx 1.41421$	$1/\Omega \approx$ **1.76322**
Arbitrary-deadline [38]	$\sqrt{2} \approx 1.41421$	2

Table .5: Summary of Upper and Lower Bounds on Speedup Factors for FP-P vs FP-NP

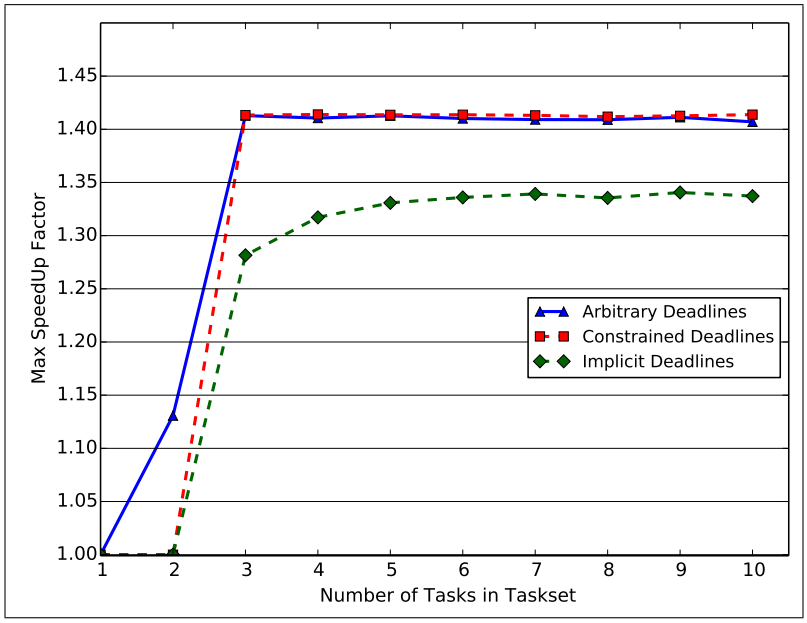

Figure .4: Speedup Factors for FP-P vs FP-NP

ID	TimePeriod	Deadline	CompTime
1	78325	78325	24387
2	98342	98342	124385
3	65706	65706	44839
4	73120	73120	32060
5	61901	61901	21304
6	59423	59423	13690
7	82547	82547	95227
8	61699	61699	1989
9	378820	378820	228
SpeedUp	**1.34059**		
U.Bound	$[\approx 1.44269]$		

Table .6: FP-P vs FP-NP for $D = T$

ID	TimePeriod	Deadline	CompTime
1	14095	10306	9580
2	101656	21134	17532
3	16117	16117	15240
4	19936	19936	17535
SpeedUp	**1.41391**		
U.Bound	$[\approx 1.41412, 2]$		

Table .7: FP-P vs FP-NP for $D \leq T$

ID	TimePeriod	Deadline	CompTime
1	2174765	2176279	3056662
2	10335794	3273652	2174428
3	2949686	3077167	2172544
SpeedUp	**1.41284**		
U.Bound	$[\approx 1.41412, 2]$		

Table .8: FP-P vs FP-NP for $D \sim T$

Conclusion

Two new lower bounds for the speedup factors were found for the arbitrary deadline cases for both preemptive and non-preemptive scheduling (see Table .3 and Table .4). These tasksets have since been verified to be correct by 3 disparate implementations of the relevant schedulability tests. Further investigation, based on the characteristics of these tasksets showed that the lower bounds are 2 and thus exact [36]. Attention is also drawn to Table .7 and Table .8 which present the tasksets that yielded the highest speedup factors for cases of FP-P vs FP-NP with constrained and arbitrary deadline respectively. The speedup values are remarkably close to the lower bounds by Davis *et al.* [38] strongly suggesting that the lower bound may be exact. This would need to be formally proved however and is therefore considered an open problem [34]. This investigation has also produced the taskset, listed in Table .6, that gives an empirical value of 1.34 for the unknown lower bound for the case of FP-P vs FP-NP with implicit deadlines. Further work is required to close the gap between this empirically derived speedup factor and the theoretical upper bound of $1/ln(2)$.

Definitions

AMC - *Adaptive Mixed Criticlaity*

AMC-max - *Adaptive Mixed Criticality - maximum*

AMC-rtb - *Adaptive Mixed Criticality - response time bound*

AMC-WH - *Adaptive Mixed Criticality - Weakly-Hard*

AMCmax-WH - *AMC-maximum with Weakly-Hard constraints*

AMCrtb-WH - *AMC-response time bound with Weakly-Hard constraints*

B_i - *Blocking Factor*

CA - *Certification Authority*

CBEDF - *Criticality Based Earliest Deadline First*

CF - *Criticality Factor*

C_i - *Worst-case execution time of a job of* τ_i

C_i^{HI} - *Worse-case execution time estimate in HI criticality mode*

C_i^{LO} - *Worse-case execution time estimate in LO criticality mode*

CP - *Criticality Probability*

CrMPO - *Criticality Monotonic Priority Ordering*

DFP - *Deadline Floor Inheritance Protocol*

D_i - *Relative deadline of task* τ_i

DMPO - *Deadline Monotonic Priority Ordering*

EDF - *Earliest Deadline First*

EDF- NP - *Earliest Deadline First - Non-Preemptive*

EDF-P - *Earliest Deadline First - Preemptive*

EDF-VD - *Earliest Deadline First with Virtual Deadlines*

ER-EDF - *Early Release - Earliest Deadline First*

FP - *Fixed Priority*

FP-NP - *Fixed Priority Non-Preemptive*

FP-P - *Fixed Priority Preemptive*

HI - *High Criticality*

HLC-PCP - *Highest-Locker Criticality Priority Ceiling Protocol*

$hp(i)$ - *Set of higher priority tasks than* τ_i

$hpHI(i)$ - *Set of tasks with higher priority than* τ_i *which are* HI *criticality*

$hpLO(i)$ - *Set of tasks with higher priority than* τ_i *which are LO criticality*

J_i - *Release Jitter of task* τ_i

L_i - *Criticality Level of task* τ_i

LO - *Low Criticality*

$lp(i)$ - *Set of tasks with lower priority than* τ_i

MC-SRP - *Mixed Criticality - Stack Resource Protocol*

MC-SRP(T) - *Mixed Criticality - Stack Resource Protocol with Thresholds*

MCS - *Mixed Criticality System*

MID - *Medium Criticality*

OCBP - *Own Criticality Based Priority*

OPA - *Optimal Priority Assignment*

PCCP - *Priority-and-Criticality Ceiling Protocol*

PCIP - *Priority-and-Criticality Inheritance Protocol*

PCP - *Priority Ceiling Protocol*

PDC - *Processor Demand Criterion*

P_i - *Priority of task* τ_i

PIP - *Priority Inheritance Protocol*

PTS - *Preemption Theshold Scheduling*

QoS - *Quality of Service*

QPA - *Quick Processor-Demand Analysis*

QPA-MC - *Quick Processor-Demand Analysis - Mixed Criticality*

R_i - *Worst-case Response time of* τ_i

RMPO - *Rate Monotonic Priority Ordering*

RTA - *Response Time Analysis*

RTOS - *Real-Time Operating System*

RTS - *Real-Time System*

SMC - *Static Mixed Criticality*

SMC-NO - *Static Mixed Criticality with No Runtime Monitoring*

SRP - *Stack Resource Protocol*

t - *Time*

τ_i - *Task i of taskset* τ

T_i - *Minimum inter-arrival time or period of task* τ_i

TWCA - *Typical Worst-Case Analysis*

U - *Utilisation value of taskset*

UAV - *Unmanned Aerial Vehicle*

UB-H&L - *Composite upper-bound on schedulability of MCS taskset*

U_i - *Utilisation value of task* τ_i

WCET - *Worst-Case Execution Time*

ZSS - *Zero Slack Scheduling*

References

[1] F. Abugchem, M. Short, and D. Xu, "A note on the suboptimality of nonpreemptive real-time scheduling," *IEEE Embedded Systems Letters*, vol. 7, no. 3, pp. 69–72, Sept 2015.

[2] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=238595&tag=1

[3] N. Audsley, "On priority assignment in fixed priority scheduling," *Information Processing Letters*, vol. 79, no. 1, pp. 39 – 44, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020019000001654

[4] N. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," in *Proceedings of IEEE Workshop on Real-Time Operating Systems and Software*, 1991, pp. 133–137. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.4438

[5] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings, "Fixed priority pre-emptive scheduling: An historical perspective," *Real-Time Systems*, vol. 8, no. 2-3, pp. 173–198, 1995. [Online]. Available: http://dx.doi.org/10.1007/BF01094342

[6] T. Baker, "Stack-based scheduling of realtime processes," *Real-Time Systems*, vol. 3, no. 1, pp. 67–99, 1991. [Online]. Available: http://dx.doi.org/10.1007/BF00365393

[7] S. Baruah and B. Chattopadhyay, "Response-time analysis of mixed criticality systems with pessimistic frequency specification," in *Proceedings of IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug 2013, pp. 237–246.

[8] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, July 2008, pp. 147–155.

[9] S. Baruah and A. Burns, "Implementing mixed criticality systems in ada," in *Reliable Software Technologies - Ada-Europe 2011*, A. Romanovsky and T. Vardanega, Eds. Springer Berlin Heidelberg, 2011, vol. 6652, pp. 174–188. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21338-0_13

[10] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 13–22. [Online]. Available: http://dx.doi.org/10.1109/RTAS.2010.10

[11] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," in *Mathematical Foundations of Computer Science 2010*, P. Hlineny and A. Kucera, Eds. Springer Berlin Heidelberg, 2010, vol. 6281, pp. 90–101. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15155-2_10

[12] S. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Algorithms - ESA 2011*, C. Demetrescu and M. Halldorsson, Eds. Springer Berlin Heidelberg, 2011, vol. 6942, pp. 555–566. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23719-5_47

[13] S. Baruah, L. Rosier, and R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, no. 4, pp. 301–324, 1990. [Online]. Available: http://dx.doi.org/10.1007/BF01995675

[14] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2011, pp. 34–43.

[15] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proceedings of 11th Real-Time Systems Symposium (RTSS)*, Dec 1990, pp. 182–190.

[16] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," *Proceedings of OSPERT*, pp. 33–44, 2010. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.182.7764

[17] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *Proceedings of 27th Euromicro Conference on Real-Time Systems (ECRTS)*, 2015, pp. 259–268.

[18] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE Transactions on Computers*, vol. 50, no. 4, pp. 308–321, Apr 2001.

[19] G. Bernat and R. Cayssials, "Guaranteed on-line weakly-hard real-time systems," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, Dec 2001, pp. 25–35.

[20] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, May 2005. [Online]. Available: http://dx.doi.org/10.1007/s11241-005-0507-9

[21] R. Bril, J. Lukkien, and W. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," *Real-Time Systems*, vol. 42, no. 1-3, pp. 63–119, 2009. [Online]. Available: http://dx.doi.org/10.1007/s11241-009-9071-z

[22] A. Burns, "The application of the original priority ceiling protocol to mixed criticality systems," *ReTiMiCS*, 2013. [Online]. Available: http://www.cs.york.ac.uk/media/computerscience/documents/researchprojects/PCPMCSRTCSA2013.pdf

[23] ——, "System mode changes - general and criticality-based," in *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), IEEE Real-Time Systems Symposium (RTSS)*, L. Cucu-Grosjean and R. Davis, Eds., 2014, pp. 3–8. [Online]. Available: http://www-users.cs.york.ac.uk/~robdavis/wmc2014/1.pdf

[24] A. Burns and R. Davis, "Adaptive mixed criticality scheduling with deferred preemption," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2014, pp. 21–30. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7010371

[25] A. Burns, M. Gutierrez, M. Aldea Rivas, and M. Gonzalez Harbour, "A deadline-floor inheritance protocol for edf scheduled embedded real-time systems with resource sharing," *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1241–1253, May 2015.

[26] A. Burns and S. Baruah, "Timing faults and mixed criticality systems," in *Dependable and Historic Computing*, C. Jones and J. Lloyd, Eds. Springer Berlin Heidelberg, 2011, vol. 6875, pp. 147–166. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24541-1_12

[27] ——, "Towards a more practical model for mixed criticality systems," in *Workshop on Mixed-Criticality Systems (colocated with RTSS)*, 2013. [Online]. Available: ftp://ftp.cs.york.ac.uk/papers/rtspapers/R:Burns:2013h.pdf

[28] A. Burns and R. Davis, "Mixed criticality systems: A review," *Department of Computer Science, University of York, Tech. Rep*, 2013. [Online]. Available: http://www-users.cs.york.ac.uk/~burns/review.pdf

[29] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages*, 4th ed. Addison Wesley, 2009.

[30] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Proceedings of 19th IEEE Real-Time Systems Symposium (RTSS)*, Dec 1998, pp. 286–295.

[31] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling*, ebook ed. Dover Publications, 2012.

[32] R. Davis and M. Bertogna, "Optimal fixed priority scheduling with deferred pre-emption," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2012, pp. 39–50.

[33] R. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1261–1276, Sept 2008.

[34] R. Davis, O. Gettings, A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "What is the exact speedup factor for fixed priority pre-emptive versus fixed priority non-pre-emptive scheduling?" in *The 6th International Real-Time Scheduling Open Problems Seminar (RTSOPS)*, July 2015. [Online]. Available: http://www.es.mdh.se/publications/3930-

[35] R. Davis, L. George, and P. Courbin, "Quantifying the sub-optimality of uniprocessor fixed priority non-pre-emptive scheduling," in *18th International Conference on Real-Time and Network Systems (RTNS)*, Toulouse, France, Nov. 2010, pp. 1–10. [Online]. Available: http://hal.inria.fr/inria-00536363

[36] R. I. Davis, A. Burns, S. Baruah, T. Rothvoss, L. George, and O. Gettings, "Exact comparison of fixed priority and edf scheduling based on speedup factors for both pre-emptive and non-pre-emptive paradigms," *Real-Time Systems*, pp. 1–36, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11241-015-9233-0

[37] R. I. Davis, T. Rothvoss, S. K. Baruah, and A. Burns, "Quantifying the sub-optimality of uniprocessor fixed priority pre-emptive scheduling for sporadic tasksets with arbitrary deadlines," *17th International Conference on Real-Time and Network Systems (RTNS)*, 2009. [Online]. Available: http://www-users.cs.york.ac.uk/~robdavis/papers/ArbitrarySpeedup2.0.pdf

[38] R. I. Davis, A. Thekkilakattil, O. Gettings, R. Dobrin, and S. Punnekkat, "Quantifying the exact sub-optimality of non-preemptive scheduling," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2015.

[39] R. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, no. 1, pp. 1–40, 2011. [Online]. Available: http://dx.doi.org/10.1007/s11241-010-9106-5

[40] R. Davis, T. Rothvoss, T., S. Baruah, and A. Burns, "Exact quantification of the sub-optimality of a uniprocessor fixed priority pre-emptive scheduling," *Real-Time Systems*, vol. 43, no. 3, pp. 211–258, 2009. [Online]. Available: http://dx.doi.org/10.1007/s11241-009-9079-4

[41] D. De Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *30th IEEE Real-Time Systems Symposium (RTSS)*, Dec 2009, pp. 291–300.

[42] M. L. Dertouzos, "Control robotics: The procedural control of physical processes." in *IFIP Congress*, 1974, pp. 807–813. [Online]. Available: http://dblp.uni-trier.de/db/conf/ifip/ifip74.html#Dertouzos74

[43] F. Dorin, P. Richard, M. Richard, and J. Goossens, "Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities," *Real-Time Systems*, vol. 46, no. 3, pp. 305–331, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11241-010-9107-4

[44] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *IEEE 34th Real-Time Systems Symposium (RTSS)*, Dec 2013, pp. 78–87.

[45] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *24th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2012, pp. 135–144.

[46] ——, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Systems*, vol. 50, no. 1, pp. 48–86, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11241-013-9187-z

[47] J. Erickson, N. Kim, and J. Anderson, "Recovering from overload in multicore mixed-criticality systems," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2015, pp. 775–785.

[48] M. S. Fineberg and O. Serlin, "Multiprogramming for hybrid computation," in *Proceedings of the Fall Joint Computer Conference*, ser. AFIPS '67 (Fall). New York, NY, USA: ACM, Nov 1967, pp. 1–13. [Online]. Available: http://doi.acm.org/10.1145/1465611.1465613

[49] T. Fleming and A. Burns, "Extending mixed criticality scheduling," in *Proceedings of Workshop on Mixed Criticality, IEEE Real-Time Systems Symposium (RTSS)*, 2013, pp. 7–12. [Online]. Available: http://www-users.cs.york.ac.uk/~robdavis/wmc/paper16.pdf

[50] ——, "Incorporating the notion of importance into mixed criticality systems," in *Proceedings of Workshop on Mixed Criticality, IEEE Real-Time Systems Symposium (RTSS)*, L. Cucu-Grosjean and R. Davis, Eds., 2014, pp. 33–38. [Online]. Available: http://www-users.cs.york.ac.uk/~robdavis/wmc2014/6.pdf

[51] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle, "Formal analysis of timing effects on closed-loop properties of control software," in *IEEE Real-Time Systems Symposium (RTSS)*, Dec 2014, pp. 53–62.

[52] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," INRIA, Research Report RR-2966, 1996, projet REFLECS. [Online]. Available: http://hal.inria.fr/inria-00073732

[53] O. Gettings, S. Quinton, and R. I. Davis, "Mixed criticality systems with weakly-hard constraints," in *Proceedings of 23rd International Conference on Real-Time Networks and Systems (RTNS)*, 2015.

[54] X. Gu, A. Easwaran, K.-M. Phan, and I. Shin, "Resource efficient isolation mechanisms in mixed-criticality scheduling," in *27th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2015, pp. 13–24.

[55] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *IEEE 32nd Real-Time Systems Symposium (RTSS)*, Nov 2011, pp. 13–23.

[56] P. K. Harter, Jr., "Response times in level-structured systems," *ACM Trans. Comput. Syst.*, vol. 5, no. 3, pp. 232–248, Aug. 1987. [Online]. Available: http://doi.acm.org/10.1145/24068.24069

[57] P. Harter, *Response Times in Level Structured Systems*, ser. University of Colorado at Boulder, Department of Computer Science. Department of Computer Science, University of Colorado, 1984. [Online]. Available: http://books.google.co.uk/books?id=fq3XtgAACAAJ

[58] H.-M. Huang, C. Gill, and C. Lu, "Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks," in *Proceedings of IEEE 18th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2012, pp. 23–32.

[59] ——, "Implementation and evaluation of mixed-criticality scheduling approaches for sporadic tasks," *ACM Transactions on Embedded Compututer Systems*, vol. 13, no. 4s, pp. 126:1–126:25, Apr. 2014. [Online]. Available: http://doi.acm.org/10.1145/2584612

[60] M. Jan, L. Zaourar, and M. Pitel, "Maximizing the execution rate of low-criticality tasks in mixed criticality system," in *Proceedings of Workshop on Mixed Criticality, IEEE Real-Time Systems Symposium (RTSS)*, 2013, pp. 43–48.

[61] M. Jones. (1997) What really happened on mars? Microsoft. http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/Mars_Pathfinder.html. [Online]. Available: http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/Mars_Pathfinder.html

[62] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986. [Online]. Available: http://comjnl.oxfordjournals.org/content/29/5/390.abstract

[63] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance [scheduling problems]," in *Proceedings of 36th Annual Symposium on Foundations of Computer Science*, Oct 1995, pp. 214–221.

[64] Y.-S. Kim and H.-W. Jin, "Towards a practical implementation of criticality mode change in rtos," Konkuk University, Korea, Tech. Rep., 2014. [Online]. Available: http://home.konkuk.ac.kr/~jinh/papers/jin_etfa14.pdf

[65] G. Koren and D. Shasha, "Skip-over: algorithms and complexity for overloaded systems that allow skips," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, Dec 1995, pp. 110–117.

[66] K. Lakshmanan, D. De Niz, and R. Rajkumar, "Mixed-criticality task synchronization in zero-slack scheduling," in *17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2011, pp. 47–56.

[67] B. W. Lampson and D. D. Redell, "Experience with processes and monitors in mesa," *ACM Communications*, vol. 23, no. 2, pp. 105–117, Feb. 1980. [Online]. Available: http://doi.acm.org/10.1145/358818.358824

[68] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Proceedings of IEEE Real Time Systems Symposium (RTSS)*, Dec 1989, pp. 166–171.

[69] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceedings of 11th IEEE Real-Time Systems Symposium (RTSS)*, 1990, pp. 201–209.

[70] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237 – 250, 1982. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0166531682900244

[71] H. Li and S. Baruah, "An algorithm for scheduling certifiable mixed-criticality sporadic task systems," in *31st IEEE Real-Time Systems Symposium (RTSS)*, Nov 2010, pp. 183–192.

[72] G. Lipari and G. C. Buttazzo, "Resource reservation for mixed criticality systems," in *Proceeding of Workshop on Real-Time Systems: the past, the present, and the future*, 2013, pp. 60–74. [Online]. Available: http://retis.sssup.it/~giorgio/paps/2013/wrts13.pdf

[73] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973. [Online]. Available: http://doi.acm.org/10.1145/321738.321743

[74] A. Massa, *Embedded Software Development with eCos*. Prentice Hall Professional Technical Reference, 2002.

[75] S. Oikawa and R. Rajkumar, "Linux/rk: A portable resource kernel in linux," in *In 19th IEEE Real-Time Systems Sumposium (RTSS)*, 1998. [Online]. Available: http://www.cs.cmu.edu/~shui/Paper/rtss98.ps.gz

[76] T. Park and S. Kim, "Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems," in *Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*. New York, NY, USA: ACM, 2011, pp. 253–262. [Online]. Available: http://doi.acm.org/10.1145/2038642.2038681

[77] M. Pilling, A. Burns, and K. Raymond, "Formal specifications and proofs of inheritance protocols for real-time scheduling," *Software Engineering Journal*, vol. 5, no. 5, pp. 263–279, Sep 1990. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=62657

[78] P. Ramanathan and M. Hamdaoui, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1443–1451, Dec. 1995. [Online]. Available: http://dx.doi.org/10.1109/12.477249

[79] B. Randell, J.-C. Laprie, H. Kopetz, and B. Littlewood, Eds., *Predictably Dependable Computing Systems (ESPRIT Basic Research Series)*, softcover reprint of the original 1st ed. 1995 ed. Springer, 2011, "ISBN-13: 978-3642797910".

[80] F. Santy, L. George, P. Thierry, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp," in *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, July 2012, pp. 155–165.

[81] F. Santy, G. Raravi, G. Nelissen, V. Nelis, P. Kumar, J. Goossens, and E. Tovar, "Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems," in *Proceedings of the 21st International Conference on Real-Time Networks and Systems (RTNS)*. New York, NY, USA: ACM, 2013, pp. 183–192. [Online]. Available: http://doi.acm.org/10.1145/2516821.2516834

[82] O. Serlin, "Scheduling of time critical processes," in *Proceedings of the Spring Joint Computer Conference*, ser. AFIPS '72 (Spring). New York, NY, USA: ACM, 1972, pp. 925–932. [Online]. Available: http://doi.acm.org/10.1145/1478873.1478995

[83] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Solutions for some practical problems in prioritized preemptive scheduling," in *IEEE Real-Time Systems Symposium (RTSS)*, 1986, pp. 181–191.

[84] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, 1990.

[85] M. Spuri, "Analysis of deadline scheduled real-time systems," Tech. Rep., 1996. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.769

[86] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 147–152.

[87] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "Quantifying the sub-optimality of non-preemptive real-time scheduling," in *25th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2013, pp. 113–122.

[88] ——, "The limited-preemptive feasibility of real-time tasks on uniprocessors," *Real-Time Systems*, vol. 51, no. 3, pp. 247–273, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11241-015-9222-3

[89] K. W. Tindell, "Extendible approach for analysing fixed priority hard real-time tasks," *Journal of Real-Time Systems*, vol. 6, 1992. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.5233

[90] K. Tindell, A. Burns, and A. Wellings, "Mode changes in priority preemptively scheduled systems," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, Dec 1992, pp. 100–109.

[91] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2007, pp. 239–243.

[92] G. Von der Bruggen, J.-J. Chen, and W.-H. Huang, "Schedulability and optimization analysis for non-preemptive static priority scheduling based on task utilization and blocking factors," in *Proceedings of 27th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2015, pp. 90–101.

[93] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 1999, pp. 328–335.

[94] H. Wyle and G. J. Burnett, "Management of periodic operations in a real-time computation system," in *Proceedings of the Joint Computer Conference*, ser. AFIPS '67 (Fall). New York, NY, USA: ACM, Nov 1967, pp. 201–208. [Online]. Available: http://doi.acm.org/10.1145/1465611.1465638

[95] C. Yao, L. Qiao, L. Zheng, and X. Huagang, "Efficient schedulability analysis for mixed-criticality systems under deadline-based scheduling," *Chinese Journal of Aeronautics*, vol. 27, no. 4, pp. 856 – 866, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1000936114001022

[96] E. Yip, M. Kuo, P. Roop, and D. Broman, "Relaxing the synchronous approach for mixed-criticality systems," in *Proceedings of Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2014, pp. 89–100.

[97] F. Zhang and A. Burns, "Schedulability analysis of edf-scheduled embedded real-time systems with resource sharing," *ACM Transactions on Embedded Compututer Systems*, vol. 12, no. 3, 2013.

[98] ——, "Schedulability analysis for real-time systems with edf scheduling," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1250–1258, 2009.

[99] Q. Zhao, Z. Gu, and H. Zeng, "Integration of resource synchronization and preemption-thresholds into edf-based mixed-criticality scheduling algorithm," in *IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug 2013, pp. 227–236.

[100] ——, "Pt-amc: Integrating preemption thresholds into mixed-criticality scheduling," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 141–146.

[101] ——, "Hlc-pcp: A resource synchronization protocol for certifiable mixed criticality scheduling," *In Proceedings of IEEE Embedded Systems Letters*, vol. 6, no. 1, pp. 8–11, March 2014.

[102] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "Flexpret: A processor platform for mixed-criticality systems," EECS Department, University of California, Berkeley, Tech. Rep., 2013. [Online]. Available: http://www.bromans.com/publ/zimmer-et-al-2014-flexpret.pdf